TRAFFIC AND RESOURCE MANAGEMENT IN ROBUST CLOUD DATA CENTER NETWORKS

Sara Ayoubi

A thesis
in
Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy
Concordia University
Montréal, Québec, Canada

July 2016

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Ms. Sara Ayoubi**

Entitled: **Traffic and Resource Management in Robust Cloud Data Center Networks**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Walaa Hamouda
_____ Chair

Dr. Jelena Mišić
_____ External Examiner

Dr. Dongyu Qiu
_____ External to Program

Dr. Roch Glitho
_____ Examiner

Dr. Lingyu Wang
_____ Examiner

Dr. Chadi Assi
_____ Thesis Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____
Dean of Faculty

Abstract

Traffic and Resource Management in Robust Cloud Data Center Networks

Sara Ayoubi, Ph.D.
Concordia University, 2016

Cloud Computing is becoming the mainstream paradigm, as organizations, both large and small, begin to harness its benefits. Cloud computing gained its success for giving IT exactly what it needed: The ability to grow and shrink computing resources, on the go, in a cost-effective manner, without the anguish of infrastructure design and setup. The ability to adapt computing demands to market fluctuations is just one of the many benefits that cloud computing has to offer, this is why this new paradigm is rising rapidly. According to a Gartner report, the total sales of the various cloud services will be worth 204 billion dollars worldwide in 2016. With this massive growth, the performance of the underlying infrastructure is crucial to its success and sustainability. Currently, cloud computing heavily depends on data centers for its daily business needs. In fact, it is through the virtualization of data centers that the concept of "computing as a utility" emerged. However, data center virtualization is still in its infancy; and there exists a plethora of open research issues and challenges related to data center virtualization, including but not limited to, optimized topologies and protocols, embedding design methods and online algorithms, resource provisioning and allocation, data center energy efficiency, fault tolerance issues and fault tolerant design, improving service availability under failure conditions, enabling network programmability, etc.

This dissertation will attempt to elaborate and address key research challenges and problems related to the design and operation of efficient virtualized data centers and data center infrastructure for cloud services. In particular, we investigate the problem of scalable traffic management and traffic engineering methods in data center networks and present a decomposition method to exactly solve the problem with considerable runtime improvement over mathematical-based formulations. To maximize the network's admissibility and increase its revenue, cloud providers must make efficient use of their's network resources. This goal is highly correlated with the employed resource allocation/placement schemes; formally known as the virtual network embedding problem. This thesis looks at multi-facets of this latter

problem; in particular, we study the embedding problem for services with one-to-many communication mode; or what we denote as the multicast virtual network embedding problem. Then, we tackle the survivable virtual network embedding problem by proposing a fault-tolerance design that provides guaranteed service continuity in the event of server failure. Furthermore, we consider the embedding problem for elastic services in the event of heterogeneous node failures. Finally, in the effort to enable and support data center network programmability, we study the placement problem of softwarized network functions (e.g., load balancers, firewalls, etc.), formally known as the virtual network function assignment problem. Owing to its combinatorial complexity, we propose a novel decomposition method, and we numerically show that it is hundred times faster than mathematical formulations from recent existing literature.

Acknowledgments

I would like to express my deepest gratitude and acknowledgment to all the great people who supported me along the way towards completing my Ph.D. studies.

First, and foremost, I am incredibly grateful to my Ph.D. supervisor Prof.Chadi Assi, for his motivation, guidance, and support. I cannot think of a more dedicated, inspiring, and caring advisor. I deeply appreciate all the knowledge and experience he bestowed upon me. I would also like to extend my deepest appreciation to my masters advisor Dr.Azzam Mourad for encouraging me to pursue a Ph.D. degree. He has truly shaped my career in a positive direction.

Moreover, I would like to thank my committee members, Prof.Roch Glitho, Prof.Lingyu Wang, and Prof.Dongyu Qiu for their time, effort, and willingness to serve on my Ph.D. committee throughout my Ph.D. studies. Thank you for making every encounter a constructive and insightful one. I would to also extend my appreciation for the external examiner Prof.Jelena Misis for her willingness to read through the thesis and serve on my defense committee.

My sincere appreciation goes to Dr.Samir Sabbah who provided his immense guidance, and taught me a lot about column-generation and the cut-and-solve based method. Thank you for your patience, and for all the insightful and enlightening discussions we had. I would also like to thank Dr.Khaled Shaban for hosting me at Qatar University during the Summer of 2014. This exchange has been an exciting and enriching learning experience.

Furthermore, I am greatly thankful to all of my colleagues in the research lab at Concordia University for providing me with a warm and friendly atmosphere. In particular, I would like to thank Mr.Yiheng Chen, Mrs.Yanhong Zhan, Ms.Hyame Alameddine, Mr.Nicolas Khoury, and Mr.Tarek Khalifa. It was a great pleasure working with you.

Last but not least, I would like to thank my family and friends for their continuous love and support. Without my mom and dad, I could not have been able to succeed throughout my life and become the person I am today. I am the product and reflection of their endless scarifies. My deepest and endless gratitude goes to my sisters for being there for me through

thick and thin; and to my grandma for being an inspiration of a powerful and dedicated woman. I would also like to thank Ms.Rima Ghanem for hosting me during my stay at Doha, Qatar; I am in debt to your kindness. Finally, I would like to thank Ms.Marylin Gerard for saving my sanity through this journey.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Cap-ex ..... Capital Expenditures

Op-ex ..... Operational Expenditures

IT ..... Information Technology

IaaS ..... Infrastructure-as-a-Service

PaaS ..... Platform-as-a-Service

SaaS ..... Software-as-a-Service

OGF ..... Open Grid Forum

IETF ..... Internet Engineering Task Force

IEEE ..... Institute of Electrical and Electronic Engineers

EC2 ..... Elastic Compute Cloud

GAE ..... Google's App Engine

CEO ..... Chief Executive Offier

NIST ..... National Institute of Standards and Technology

N/S ..... North-South

VM ..... Virtual Machine

E/W ..... East-West

VNE ..... Virtual Network Embedding

SVNE ..... Survivable Virtual Network Embedding

DC ..... Data Center

DCN ..... Data Center Network

IP ..... Internet Protocol

STP ..... Spanning Tree Protocol

VLAN ..... Virtual Local Area Network

VN ..... Virtual Network

CPU ..... Central Processing Unit

VNM ..... Virtual Node Mapping

VLM ..... Virtual Link Mapping

k-SP ..... k-Shortest Paths

MCF ..... Multi-Commodity Flow

MVNE ..... Multicast Virtual Network Embedding

QoS ..... Quality of Service

SVN ..... Survivable Virtual Network

SLA ..... Service Level Agreement

SDN ..... Software Defined Network

NFV ..... Network Function Virtualization

IDS ..... Intrusion Detection System

VNF ..... Virtual Network Function

ILP ..... Integer Linear Program

MVN ..... Multicast Virtual Network

DP ..... Dynamic Programming

ProRed ..... Prognostic Redesign

RELIEF ..... Reliable Embedding Framework

MSTP ..... Multiple Spanning Tree Protocol

SPAIN ..... Smart Path Assignment in Networks

CP ..... Constraint Programming

ECMP ..... Equal Cost Multi-Path

LP ..... Linear Program

CG ..... Column Generation

RC ..... Reduced Cost

Gbps ..... Gigabit per second

GB ..... GegaByte

RAM ..... Random Access Memory

HPC ..... High Performance Computing

DFS ..... Distributed File-Systems

MVNE-H .... Multicast Virtual Network Embedding Heuristic

MVNE-HNM Multicast Virtual Network Embedding Heuristic with Node Mapping Model

DFS ..... Depth First Search

BFS ..... Breadth First Search

AS ..... Aggregation Switch

CS ..... Core Switch

MVNE-HFF . Multicast Virtual Network Embedding Heuristic with First First Node Mapping

MVNE-HG .. Multicast Virtual Network Embedding Heuristic with Greedy Node Mapping

NP ..... Nondeterministic Polynomial Time

CMP ..... Cloud Management Platform

EVN ..... Enhanced Virtual Network

APG ..... Auxiliary Protection Graph

BG ..... Backup Group

WG ..... Working Group

VMP ..... Virtual Machine Placement

SVNE-M ..... Survivable Virtual Network Embedding Model

SVNE-H ..... Survivable Virtual Network Embedding Heuristic

REM ..... Restoration Model

MVNR ..... Multicast Virtual Network Restoration

ST ..... Steiner Tree

REAL ..... Restoration Algorithm

ToR ..... Top-of-Rack

SR ..... Server Rack

ARES ..... Availability-Aware Reconfiguration Scheme

JENA ..... Just-Enough Availability

MTBF ..... Mean Time Between Failure

MTTR ..... Mean Time To Repair

MINLP ..... Mixed Integer Non-Linear Program

GQAP ..... Generalized Quadratic Assignment Problem

MRI ..... Migration/Redundancy Iterator

SG ..... Static Greedy

DG-P ..... Dynamic Greedy with Protection

DG-NP ..... Dynamic Greedy without Protection

FT ..... Fat Tree

HIVI ..... High-availability Virtual Infrastructure

WAN ..... Wide Area Network

SFC ..... Service Function Chaining

IETF ..... Internet Engineering Task Force

IDS ..... Intrusion Detection System

NF ..... Network Function

MIQCP ..... Mixed Integer Quadratically Constrained Program

Chapter 1

Introduction

1.1 Thesis Statement

In the last decade, Cloud Computing has gained momentum for enabling the dream of computing as a "utility" [1]. Very much like every day utilities (e.g., water, electricity, gas, telephony, etc.), the details are abstracted to the users [2]. Clients are unaware of where the service is located and how it is delivered to them; they simply use as much as they need and are billed according to their usage. Indeed, this pay-as-you-go business model offers many salient features. It greatly reduces Capital Expenditures (Cap-ex), by providing the illusion of infinite network resources. Today, companies no longer need to invest upfront in dedicated networks, or over-provision in hardware resources for peak loads, but rather resources can now be leased and released on-demand to match the user's actual needs over time. This *cost-associativity* feature is particularly interesting for companies with large batch-oriented tasks; since now the cost of using 1000 servers for 1 hour is equivalent to the cost of using 1 server for 1000 hours [1]. Further, it also provides savings in terms of Operational Expenditures (Op-ex) by delegating the responsibility of hardware provisioning/configuration and other Information Technology (IT) related operations to a dedicated entity. In this regard, many companies are embracing cloud computing services to improve the scale, cost-effectiveness, and reliability of their operations. This includes organizations in the commercial, private, governmental, and defense sectors. Now, as this transition unfolds, various cloud deployment models have emerged to support different client applications, i.e., Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), see [3]. In addition numerous cloud standardization efforts are also taking shape within the Open Grid Forum (OGF) [4], [5], Internet Engineering Task Force (IETF) [6], and Institute of Electrical and

Electronic Engineers (IEEE) [7]. Finally, the broader IT sector is also seeing the emergence of many cloud service provider organizations [1]. For example, Amazon is offering its Elastic Compute Cloud (EC2) service to build resizable computing facilities for web-scale applications. Meanwhile Google's App Engine (GAE) also provides on-line application development/hosting environments and Microsoft's Azure platform service supports scalable application development. Other cloud offerings include SmartCloud by IBM and the open-source Nimbus framework [3].

The notion of cloud computing is not a new one; John McCarthy has envisioned it in 1960s [8]. However, it is only until 2006 that the term "cloud computing" really gained popularity when coined by Google's CEO. Despite being used in the title of many conferences (e.g., CloudNet, CloudCom, CLOUD, etc.), and having dedicated journals (International Journal of Cloud Computing, IEEE Transactions on Cloud Computing, etc.), there has been significant ambiguities around the definition of "cloud computing" [8]. This is mainly due to the fact that cloud computing is not a new technology, but rather a new business model that combines and leverages existing technologies. With the goal of alleviating confusions and offering a standard definition, the National Institute of Standards and Technology (NIST) published the following definition [9] of Cloud Computing: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.* Despite not being a novel notion, the promise of computing-as-a-utility has only been made possible due to the recent advancements in processing, storage, and high-bandwidth networking technologies, which have rendered computing resources more powerful and ubiquitously available. This has led to the emergence of mega-data centers constructed at low-cost locations, thereby allowing cloud providers to offer services below the cost of ownership, while also making a profit. Further, the emergence of network virtualization enabled the mutli-tenancy concept, and allowed cloud provider to make efficient use of their network resources via consolidation; as opposed to running tenant services on dedicated machines. The latter is particularly attractive, given the low average server utilization [10] (typically less than 20%).

With the rapid rise and adoption of cloud computing, data centers are growing at unprecedented scale. Data collected in 2011 on Google's energy consumption suggests that it probably uses 900,000 servers [11], Amazon has at least 30 data centers, each housing between 50 to 80 thousand servers [12]. In 2013, Microsoft's CEO reported the scale of

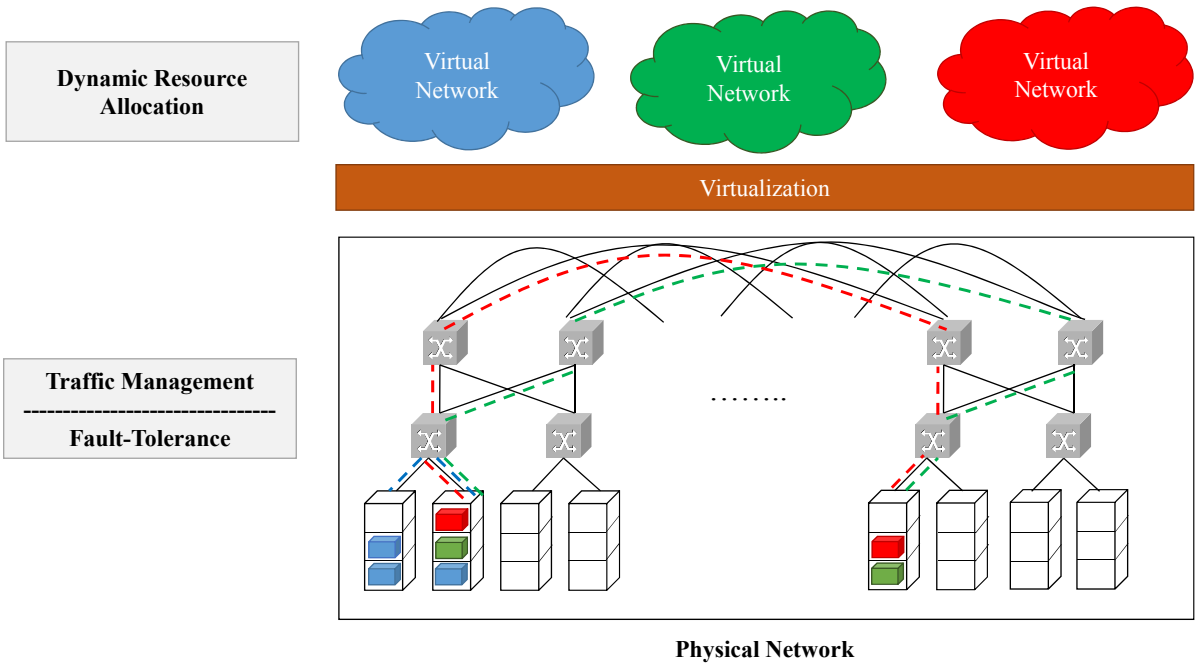


Figure 1.1: Effects of Cloud Computing on Data Center Networks

Microsoft's data centers at 1 million servers [13]. Along with the emergence of mega cloud data centers, many challenges also surfaced in enabling its basic functionalities [14]. For instance, the traditional hierarchical data center network topologies [15] were mainly designed with North-South (N/S) traffic patterns in mind to support client/server applications. However, with virtualization, the applications environment has shifted towards service-delivery with a lot of Virtual Machine-to-Virtual Machine (VM-to-VM) communication [16]; i.e., East-West (E/W) traffic. This shift in traffic patterns has attracted the attention of the research community and industry alike towards developing novel traffic management schemes and proposing alternative network architectures [17–24]. Traffic management in cloud data center networks is one of the problems that this thesis addresses. Further, the freedom to collocate multiple tenants over the same physical infrastructure raises the question of how to efficiently allocate network resources among the various tenants [14]; formally known as the Virtual Network Embedding (VNE) problem. This thesis undertakes a variation of the VNE problem. Moreover, in a virtualized infrastructure where multiple virtual networks (or tenants) are running atop the same physical network (e.g., a data center network, as shown in Figure 1.1), a single failure can bring down multiple services, leading to millions of dollars in penalty cost. It is estimated that a single minute of downtime can cost an average of 7,900$ [25], not to mention jeopardizing the reputation of the infrastructure provider. Indeed, business continuity/service availability is listed among the top 10 obstacles [1] that

inhibit many organizations from transitionning to the cloud. This fear is completely justifiable given the failure-prone nature of cloud data center networks [26, 27]. Hence, attention converged towards solving the Survivable Virtual Network Embedding (SVNE) problem, and this thesis joins these efforts.

In summary, this thesis addresses the problems of traffic management and resource allocation in cloud data centers, as illustrated in Figure 1.1. The traffic management problem is considered in the case of policy-aware and policy-oblivious network flows. Further while addressing the resource allocation problem, this thesis takes a practical outlook by considering the different communications modes that cloud services may exhibit, as well as the elastic nature of these services (scale up/down over time), while accounting for the failure-prone nature of cloud data center networks. In the following section, we motivate each of these problems that outline the main contributions of this dissertation.

1.2 Problem Motivation

1.2.1 Traffic Management in Cloud Data Center Networks:

Data Center networks are a vital element of the cloud computing paradigm; they represent the infrastructure layer that supports and sustains the various cloud-based applications. A Data Center (DC) is a large, dark, and cold facility housing hundreds of thousands of servers interconnected together; and the DC Network (DCN) represents the network architecture and protocols governing a DC [28]. The conventional data center network topology [15] consists of a 3-layered architecture (as illustrated in Figure 1.2): the lower layer contains the server racks with a top-of-rack switch to allow intra-rack communication. The second layer consists of aggregate switches that enable data exchange between servers in different racks. The aggregation switches are then connected to the top layer core routers that carry traffic from the data center to the Internet. This architecture is known to present a multitude of limitations and drawbacks: namely, due to the use of expensive routing hardware, the current topology adopts a scale-up strategy; hence, as traffic moves towards the top layers switches/routers, the level of oversubscription increases. This problem is particularly aggravated with the emergence of network virtualization that shifted the dominant traffic pattern from north-south to east-west flows. Indeed, a study led by Microsoft research [17] has shown that the level of oversubscription in canonical data centers can reach a factor of 1:240. Further, the core layer, also known as Layer-3 network, imposes the use of hierarchical Internet Protocol (IP) addresses which hinders services mobility (with the exception of

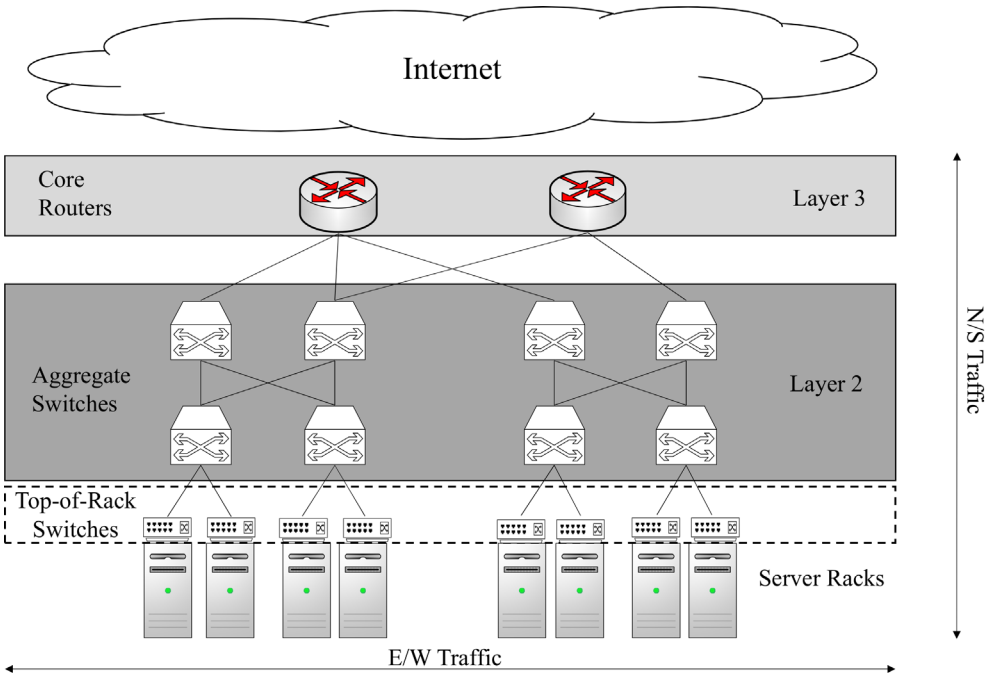


Figure 1.2: Canonical DCN Topology

mobile IP) in the data center, since it requires re-adjusting the IP addresses after migration. Such a procedure can be cumbersome and usually requires human intervention to reconfigure the routers. Moreover, the scale-up structure of the canonical DCN topology renders lower-layer network components vulnerable to any failure at the higher-layers; e.g., failure of a top-of-rack switch will disconnect the entire server rack from the rest of the network.

In light of the above, DCN topologies have received significant attention from research and academia alike [17, 19, 23, 29, 30], due to the impact the former has on the agility and reconfigurability of DC networks [31]. A take away from these approaches is the unanimous appraisal for a layer-2 network topology, particularly Ethernet-based data center networks. Layer-2 data center networks offer better manageability and application mobility. Ethernet is attractive for being ubiquitous, inexpensive, and off-the shelf commodity; it therefore enables scale-out network designs to eliminate network oversubscription and support the east-west traffic pattern of the next-generation DCN. Further, it allows the hosted services to move around in the network seamlessly without any service disruption. This capability plays a fundamental part in enabling better load balancing and fault tolerance. For instance, if one part of the network is overloaded, services can migrate to a less congested part of the network. Services migration also allows overcoming hardware or link failures, by migrating the applications hosted on the affected node to other stable nodes.

One limitation of Layer-2 DCN is the inherent forwarding protocol in Ethernet switches. The

conventional traffic splitting technique in Ethernet switches is the Spanning Tree Protocol (STP). STP consists of finding a loop-free path that spans all the nodes in the network; thereby concentrating all the traffic in the network through a limited subset of substrate links, rendering the rest of the physical links in the network unutilized. This inconvenience is particularly noticeable in multi-layered network topologies that exhibit multiple equal cost paths between every pair of nodes (e.g., FatTree DCN topology [28]). A promising alternative is the use of Virtual Local Area Networks (VLANs); where each VLAN will have its own STP mapped to different paths for better traffic distributions/balance. However, finding the optimal traffic split between VLANs is the well-known NP-Complete VLAN assignment problem [19]. The size of the search space of the VLAN assignment problem is huge, even for small size networks. To this extent, Chapter 2 of this dissertation is dedicated to study the VLAN assignment problem in Layer-2 DCN.

1.2.2 The Virtual Network Embedding Problem

Conventionally, hosting applications in data centers was performed in a dogmatic approach, where each application would run on a dedicated set of servers. This approach leads to poor resource utilization, particularly since most of the servers might be highly under-utilized, and mostly provisioned to handle sudden load surges. In fact, it has been shown [32] that the average server utilization in a typical data center is 10-15% of its full capacity. Server virtualization techniques (e.g. Hypervisors) circumvent these limitations by allowing multiple tenant applications to co-exist on the same physical server. This approach greatly enhances the underlying network utilization and allows resource consolidation. In addition to server virtualization, tenants in a cloud data center usually demand bandwidth guarantees to achieve performance predictability of their application. Here, each tenant's application is abstracted as a Virtual Network (VN), comprising of the VMs running the tenant's service and virtual links representing the communication pattern between these VMs. The challenge therefore becomes to find the optimal strategy to map these virtual networks to the substrate network, such that the utilization of the underlying infrastructure is maximized. This problem is the well-known Virtual VNE problem, and has been proven [33] to be NP-Hard owing to the bundle of factors and constraints that emerge with it.

1.2.2.1 Problem Definition

The VNE problem is the main resource allocation problem in network virtualization that indicates the optimal placement of VNs onto the substrate network, such that desired design

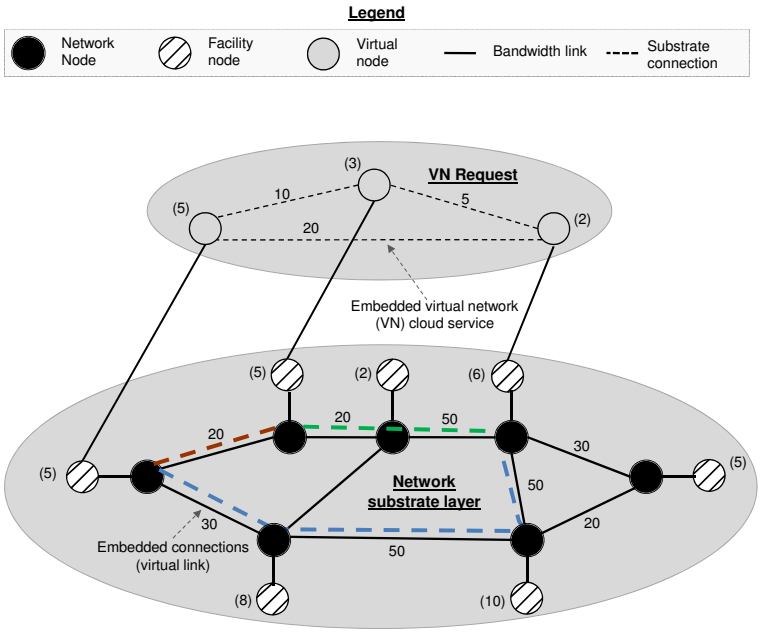


Figure 1.3: Virtual Network Embedding Problem

objectives are achieved (minimize VNs blocking rate, maximize substrate network's acceptance ratio, maximize long-term profit, etc.). Below, we formally define the VNE problem:

1. **The Substrate Network:** A substrate network represents the physical infrastructure (e.g., DCN) that hosts multi-tenant applications. In the VNE problem, the network is represented as an undirected graph, denoted by $G^s = (N,L)$; where N represents the set of substrate nodes, and L is the set of substrate links. Here, we distinguish between two types of physical nodes (as shown in Figure 1.3); facility nodes (server racks) that host virtual machines, and network nodes that assume the role of routing/forwarding traffic in the substrate network. We denote $\bar{N}$ as the set of facility nodes, and $\hat{N}$ as the set of network nodes; where $N = \bar{N} \cup \hat{N}$. Each facility node $n \in \bar{N}$ is associated with R resource types (Central Processing Unit (CPU), memory, etc.), each with a finite capacity denoted by c_n^r. Similarly, a substrate link $l \in L$ has bandwidth capacity, denoted by b_l.

   Figure 1.3 illustrates a substrate network, where the capacity of the facility nodes and physical links is represented by the number next to each node and link, respectively. For the sake of clarity, we only show a single resource type for the facility nodes.

2. **The VN Request:** A virtual network represents a client's request to deploy an application in a cloud data center. There exists several abstraction models for VNs, the pipe

model, hose (and hierarchical) model, and tenant application graph are the three most widely used representations [34]. In this dissertation, we adopt the pipe model with bandwidth guarantees to provide an assurance on the requested bandwidth between every pair of VMs. A VN consists of a set of virtual nodes connected through virtual links, denoted by $G^v = (V,E)$. Virtual links are used to describe the communication pattern among the virtual nodes, and each virtual link requires a specific amount of bandwidth, denoted by b'_e. In addition, each virtual node is usually associated with resource demand of c'^r_v for each resource type $r \in R$ (CPU, memory, etc).
We illustrate an example of a VN request in Figure 1.3. This VN request consists of 3 virtual nodes connected by 3 virtual links. Note that the resource demands are represented by the numbers above the virtual nodes and links. For the sake of clarity, we only show a single resource type for the virtual nodes.

3. **The VNE Problem:** When a VN request arrives, the network operator needs to decide whether to accept the VN request or not, and if accepted where to place that VN, such that enough network resources are allocated to satisfy its demands, without violating the capacity constraints of the substrate network. The VNE problem can be logically divided into two subproblems: Virtual Node Mapping (VNM) and Virtual Link Mapping (VLM). Figure 1.3 illustrates the VNM and VLM subproblems. The VNM solution indicates the physical substrate nodes onto which the virtual nodes will be embedded. Each physical node must be able to assume the resource demands of the embedded virtual node(s). The mapping of virtual links onto the substrate network is translated into a path, or a set of paths, that traverse one or more physical links. The VNE problem can be formulated as follows:

   Problem Definition 1.1. *Given a substrate network $G^s = (N, L)$ and a VN request $G^v = (V, E)$, Find an optimal mapping $M = (M_N, M_L)$ of the VN request onto the substrate network, such that the demands of the virtual nodes and virtual links are satisfied, without violating the capacity of the substrate network.*

   Note that a mapping M holds the solution for the two subproblems:

   (1) Virtual Node Mapping (VNM): M_N: $V \longrightarrow N$

   (2) Virtual Link Mapping (VLM): M_L: $E \longrightarrow P$; where P represents a path, (or a set of paths) in the substrate network.

1.2.2.2 Literature Review

The VNE problem has attracted a lot of attention from the research community [1], as a result, a handful of optimization- and heuristic-based embedding strategies have been proposed. Here, most optimization schemes try to minimize substrate usages or maximize revenues (minimize blocking) for a-priori demands. Some also aimed to reduce energy usage through resource consolidation [36], [37]. However, since many of these formulations are very complex even for moderate network sizes, various relaxations have also been proposed [38–46]. In general, these heuristics can be classified into two types: separate node/link mapping (two-stage) and joint node/link mapping (single-stage) strategies. Two-stage algorithms first compute a subset of storage/computing nodes to satisfy the resource requirements of the VN (greedy approach) and then route loop-free virtual link connections to meet the link and/or delay constraints, e.g., via shortest path [40], k-shortest paths (k-SP) [43], or multi-commodity flow (MCF) routing [43]. Note that MCF schemes assume path-splitting at the network substrate nodes/ switches and may lead to packet stream reordering. Nevertheless, the separate mapping of VN nodes and links generally lacks resource coordination (across the VN topologies) and can lead to high resource inefficiencies [44]. Unified single-stage heuristics have been developed to jointly map VN nodes/links and improve resource efficiency or lower the blocking rates [45], [44]. Furthermore, some researchers have also looked at VN migration to re-optimize allocations under time-varying dynamic demands [43], [46].

A main weakness in these suggested approaches, in addition to the lack of a guarantee on the quality of the obtained solution, is that most of the existing work does not characterize the mode of communication in the VN requests, assuming that they all exhibit unicast or one-to-one communication only. When in fact, depending on the type of service running in these VNs, communication among the participating VMs can be either unicast, multicast (one-to-many) or broadcast (all-to-all). Characterizing the type of communication in VNs is crucial for achieving optimal network operation. In unicast communication, the sender transmits data to a single receiver; thus, handling multicast communication as unicast requires transmitting multiple copies of the same data to each receiver. On the other hand, multicast routing transmits a single copy of the same data to all the receivers in a single multicast communication group. Hence, routing multicast and broadcast flows as unicast creates redundant traffic which decreases the network's throughput and leads to bottlenecks, particularly if the network contains many conflicting paths.

[1] A cohesive survey on the VNE problem with a thorough taxonomy of the various contributions in the literature can be found in [35].

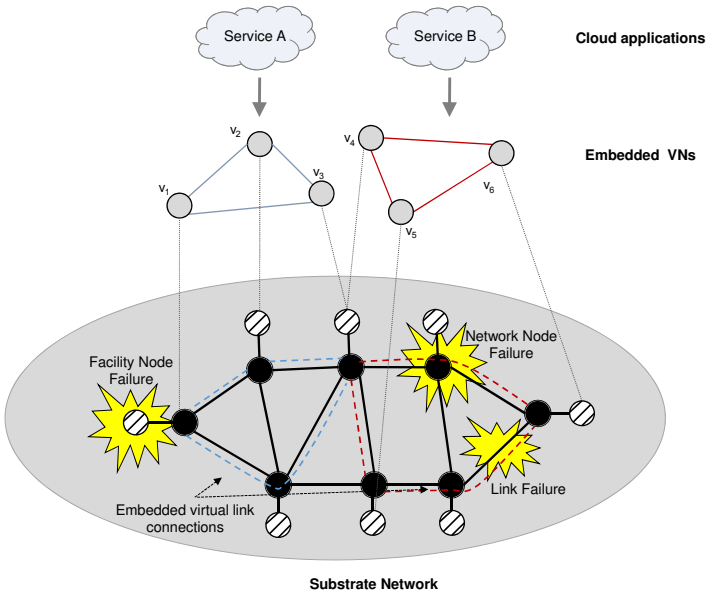


Figure 1.4: Failure in the Substrate Network

In light of the above, Chapter 3 of this dissertation considers the VNE problem for services with one-to-many communication modes; or what we refer to as Multicast VNE (MVNE) problem. Here, we list the various cloud applications and services that benefit from multicast to disseminate their traffic, and we highlight their unique properties and distinct Quality of Service (QoS) requirements, most notably the end-delay and delay-variation constraints. Further, we showcase the limitations of handling a multicast VN as unicast. This therefore motivates the need for a dedicated study, and a tailored multicast VNE scheme that responds to these unique properties.

1.2.3 The Survivable Virtual Network Embedding Problem

Another limitation of the existing literature for the VNE problem is that they all assume that the substrate network is available at all times. When in fact, failure in the physical infrastructure is common due to a multitude of reasons [47] that can affect one or many network components. Figure 1.4 illustrates the different types of failures that can occur in a data center network. We observe 3 types of failures :

1. **Facility Node Failure:** A facility node failure brings down the physical node itself, as well as, all hosted VMs. Hence, a single facility node failure can potentially lead to multiple virtual machines failure. Figure 1.4 shows an example of a facility node failure that leads to the failure of virtual node v_1.

2. **Physical Link Failure:** A physical link failure disconnects its respective edge nodes, as well as, all the virtual links whose corresponding physical paths are routed through it. Thus, a single link failure can potentially lead to multiple virtual links failure. Figure 1.4 shows that the failure of a single physical link disconnects the virtual link connecting v_5 and v_6, since it is mapped onto a physical path that traverses the failed link.

3. **Network Node Failure:** A network Node failure brings down the network node that handles the switching/routing of traffic in the substrate network. Network node failure is coupled with both facility node and physical node failures as illustrated in Figure 1.4. We observe that the failure of a single network node disconnects its incident facility node from the rest of the network (which would disconnect any VM mapped onto that facility node). Subsequently, the physical links that are connected to this failed network node can no longer route the traffic through it.

Given the failure-prone nature of DCN, attention converged towards solving the survivable VNE [48–55] to achieve reliability and fault-tolerance, and reduce the incurred penalties caused by failures. The SVNE problem consists of mapping virtual network requests, while providing protection or recovery mechanisms against failures in the substrate network. Survivability features can either be integrated at the VN side, or at the substrate network level: at the VN side, survivability is achieved by redesigning the VN request into a Survivable Virtual Network (SVN), this can be done by augmenting the VN with backup nodes and then mapping the resultant SVN to the physical network. At the physical level, survivability can be embedded as VN requests arrives, where the VN request is first embedded, and then backup nodes/links are explored in a reactive/proactive manner. The reactive approach consists of finding backups once a failure occurs, to assume the failed resource(s). While the proactive approach, pre-computes the backups for critical resources in a VN, upon arrival (on-demand). Another approach to incorporate survivability at the physical level is known as "pre-configured", and consists of setting up the substrate network with backups beforehand, prior to receiving any VN request (pre-allocated). The difficulty of the SVNE problem resides in finding the optimal tradeoff between the level of protection provided for each VN request, and the utilization efficiency of the substrate network. Given its complex nature, most of the relevant literature focused on designing heuristic-based algorithms, and restricting the problem space to a single type of failure; most of them further simplify the problem by considering that only a single network component would fail at any given point in time.

1.2.3.1 The Survivable Virtual Network Redesign Problem

When a facility node fails, the hosted virtual node(s) needs to migrate to a backup facility node, as well as its associated connections to other virtual nodes belonging to the same VN. As we have previously mentioned, one way of achieving this failure recovery is by redesigning the VN request into a SVN, and then mapping the resultant SVN onto the physical network. This redesign consists of augmenting the original VN with backup nodes. Each backup node is in charge of protecting one or many primary nodes. Hence, backup virtual links must be established between each backup node and the neighbors of the primary nodes it protects. Further, given that a single failure might occur at any point in time, backup-sharing [53] can be employed to alleviate some of the backup footprint. This is particularly useful given that the provisioned backup resources remain idle until failure occurs.

The survivable redesign technique encloses multiple challenges. Among these challenges is deciding how many backup nodes to use and how to allocate these backup nodes to the primary nodes in each VN, such that we minimize the amount of reserved resources in the substrate network. This problem is of paramount importance since these provisioned resources will remain idle until failures occur. Hence, over-provisioning can greatly impact the network's ability to admit future requests. However, in most of the existing work, designing an SVN is limited to a fixed number of backup nodes; further backup-sharing is only explored and optimized during the embedding phase. This renders the existing redesign techniques agnostic to the backup resource sharing in the substrate network, and highly dependent on the efficiency of the adopted mapping approach. In Chapter 4 of this dissertation, we diverge from this dogmatic approach by looking at the problem of finding the minimal number of redundant nodes that promotes backup sharing in the substrate network.

1.2.3.2 Restoration Methods for Cloud Multicast Virtual Networks

As mentioned before, many applications and services hosted in cloud data center networks today rely on multicast communication to disseminate traffic. Hence, the existing survivable embedding schemes fail to cater to the distinctive properties and QoS requirements that multicast services entail. In this regard, we devote Chapter 5 of this thesis to investigate the impact of failure on multicast services residing in data center networks.

1.2.3.3 A Reliable Embedding Framework for Elastic Virtualized Services in the Cloud

While survivability analysis makes an implicit assumption that a failure will occur, availability assumes that failures are somewhat random and it is the role of the infrastructure provider to ensure that each service is supported by robust hardware components [56]. These two metrics are highly complementary, since survivability guarantees failure recovery in the event of any failure, and is particularly useful for mission critical services that do not tolerate failures (e.g., banking systems), or to avoid single points of failure (e.g., a middle node in a hub and spoke network). Whereas availability-aware embedding guarantees a ratio of service uptime over the total elapsed time for the tenant's service, typically negotiated in the form of a Service Level Agreement (SLA). If the service's availability drops below a certain threshold, the infrastructure provider will incur a monetary penalty that will be disbursed to the affected tenants. E.g., AmazonEC2 [57] provides a 10% service credit for any service whose monthly availability drops below 99.95%, and a 30% credit if it goes under 99%.
In this thesis, we look at the problem of availability-aware VNE. Existing availability-aware embedding schemes [58, 59] overlooked the "availability overprovisioning" problem; that is providing a service with more availability than requested. As this manuscript will show, such approach greatly impacts the network's admissibility. Further, earlier work assumed tenants's requests to be static; as-in the tenants's resource demands do not change throughout the lease period. When in fact, it has been shown that 90% of IT services exhibit periodic resource demands [60]. As tenants scale their services, the cloud provider not only needs to adapt the allocated resources to meet the requested changes, but also tune the initially devised failure mitigation plan for each scaling tenant. To this extent, in Chapter 6 of this dissertation, we study the problem of VNE with "just-enough" availability, as well as the problem of maintaining the availability guarantee as these services scale during their residency. To the best of our knowledge, this work is the first to address the problem of managing reliable elastic VNs in the cloud.

1.2.4 Towards Enabling the Support of Network Programmability:

Enabling DCN programmability is a recent trend that emerged to manage the network programmatically. This goal is driven by the need to avoid box-to-box configuration, gain better end-to-end traffic control, and overall enhance and facilitate the management of mega data centers. Recently, Software Defined Networks (SDNs) has been receiving lots of attention

both in academia and more in the industry. This is mainly due to its ability to enable network control to become completely programmable by decoupling it from the data plane, thereby removing the intelligence from the hardware performing the routing/switching that typically use proprietary firmware. Another trend along these efforts is the emergence of softwarized network functions, also known as Network Function Virtualization (NFV) [61]. NFV emerged to tackle the inconvenience of hardware middleboxes (beyond routing/switching). Middleboxes (e.g., load balancers, firewalls, Intrusion Detection Systems (IDS)) have gained popularity due to the significant value-added services these network elements provide to traffic flows, in terms of enhanced performance and security. Policy-aware traffic flows usually need to traverse multiple middleboxes in a predefined order to satisfy their associated policies, also known as *Service Function Chaining*. Typically, Middleboxes run on specialized hardware, which make them highly inflexible to handle the unpredictable and fluctuating-nature of traffic, and contribute to significant Cap-ex and Op-ex to provision, accommodate, and maintain them. NFV is a promising technology with the potential to tackle the aforementioned limitations of hardware middleboxes. Yet, NFV is still in its infancy, and there exists several technical challenges that need to be addressed, among which, the Virtual Network Function (VNF) assignment problem tops the list. The VNF assignment problem stems from the newly gained flexibility in instantiating VNFs (on-demand) anywhere in the network. Subsequently, network providers must decide on the optimal placement of VNF instances which maximizes the number of admitted policy-aware traffic flows across their network. Existing work consists of Integer Linear Program (ILP) models, which are fairly unscalable, or heuristic-based approaches with no guarantee on the quality of the obtained solutions. Hence, the need for an approach that is more scalable than ILP-based formulations while also providing exact solution to this difficult problem. To this extent, we dedicate Chapter 7 of this dissertation to propose such an approach.

1.3 Thesis Contributions

The main contributions of this thesis can be summarized as follows:

- We introduce a novel decomposition approach to solve the VLAN mapping problem in cloud data centers through column generation. Column generation is an effective technique that is proven to reach optimality by exploring only a small subset of the search space. We introduce both an exact and a semi-heuristic decomposition with the objective to achieve load balancing by minimizing the maximum link load in the network.

Our numerical results show that our approach explores less than 1% of the available search space, with an optimality gap of at most 4%. We also compare and assess the performance of our decomposition model and state of the art protocols in traffic engineering. This comparative analysis shows that our model attains encouraging gain over its peers.

- We formally define the VNE problem for Multicast VNs (MVNs) and prove its NP-Hard nature. We propose a novel 3-Step approach for solving the MVNE problem, and introduce the receivers embedding problem over multicast trees. We mathematically formulate the receivers embedding problem, and propose a Dynamic Programming (DP) approach for MVNs with homogeneous resource demands, that is solvable in polynomial-time over multicast trees with constant nodal degree. For tree-like data center network topologies, we prove that our 3-Step MVNE with the DP for receivers embedding provides optimal solution in polynomial-time for MVNs with homogeneous resource demands. Finally, we propose a Tabu-based search for solving the MVNE problem for multicast services with heterogeneous resource demands over arbitrary network topologies. We compare our Tabu approach against the 3-Step MVNE and other embedding heuristics, using multiple metrics and over various substrate networks. Our numerical results show that our Tabu-based search yields high network admissibility in considerably fast runtime.

- We study the SVN design problem, and we argue that the existing literature's proposition of fixing the number of backup nodes could yield infeasible or even costly mapping solutions, and we provide several motivational examples to support this claim. Subsequently, we introduce ProRed, a novel prognostic redesign technique that promotes the backup resource sharing at the virtual network level, prior to the embedding phase. We compare our proposed method against existing redesign techniques, and we show that our approach achieves lower-cost mapping solutions and greatly enhances the achievable backup sharing, boosting the overall network admissibility.

- We study the impact of failure on multicast services residing in data center networks. Through motivational examples, we draw the observation that mending failures of multicast services not only requires failure restoration, but also maintenance to preserve the QoS requirements of the distribution tree. We focus the scope of this work on the case of facility node failure, and we mathematically formulate the problem and prove its NP-Complete nature. Further, we prove that the problem of restoring multicast VNs

against facility node failure can be solved in polynomial-time when applied to multi-rooted tree-like data center network topologies. We evaluate our proposed restoration schemes for typical DCN topologies against a Greedy and a Steiner-based restoration schemes, and we show that our suggested method outperforms its peers in terms of restoration ratio and total achievable revenue.

- We consider the problem of managing scaling requests for services with strict availability, and we propose RELIEF: a novel reliable embedding framework for elastic services in data center networks. RELIEF consists of two main modules: a cost-efficient availability-aware resource allocation (embedding) module for incoming tenants, and a reliable reconfiguration module to adapt the embedding of hosted services as they scale. We compare our proposed framework against peer and benchmark algorithms, and we show that our proposed framework enhances network's admissibility, and in return increases the cloud provider's long term revenue, compared to peer and benchmark algorithms.

- Finally, we propose a novel *Cut-and-Solve* based approach to solve the VNF assignment problem for policy-aware traffic steering. Our cut-and-solve approach jointly addresses the problem of VNF placement and policy-aware traffic steering to maximize the number of flows routed across the network. We compare our approach against an ILP-based formulation and a heuristic method, and we show that our approach achieves the optimal solution (as opposed to heuristic-based methods) 700 times faster than the ILP.

1.4 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 addresses the VLAN assignment problem in Layer-2 DCN. In Chapter 3, we discuss the multicast virtual network embedding problem and present our first contribution in this area. Chapters 4-6 are dedicated for tackling the reliable resource allocation problem; where Chapters 4 addresses the problem from a survivability point of view, Chapter 5 is concerned with the restoration of multicast services, and Chapter 6 focuses on the availability-aware VNE problem for elastic services. Finally in the efforts to support network programmability, Chapter 7 is dedicated to propose a novel cut-and-solve based approach to solve the VNF assignment problem. We conclude this manuscript in Chapter 8 and provide future directions for this research. It is important

to note that throughout this dissertation, figures are enumerated relatively to each chapter.

Chapter 2

Towards Scalable Traffic Management in Cloud Data Centers

2.1 Problem Statement

In the last decade, significant attention has been devoted towards rethinking the data center network architecture and topologies [17–19,23,29,30]; this was in no small portion due to the vital role data centers play in supporting the multi-million dollar industry of Cloud Computing. While many praised Ethernet for being the ideal technology for intra-data center networks, this technology is difficult to scale as data centers grow in size and load. Indeed, Ethernet is attractive for being ubiquitous, inexpensive, and off-the shelf commodity. Its ease of use, self-learning, and independent forwarding capability makes it ideal for data center network environment. The inherent traffic splitting technique in Ethernet switches is the Spanning Tree Protocol (STP). STP consists of finding a loop-free path that spans all nodes in the network. Links that violate the loop-free constraint are excluded by blocking the corresponding switch ports. To achieve all these features, Ethernet relies on broadcast-based communication and packet flooding to learn host location, which makes it highly unscalable. Moreover, the STP protocol suffers from poor resource utilization [18] as it fails to exploit the path redundancy in the physical topology, rendering many links in the network underutilized.

A promising alternative is the use of Virtual Local Area Networks (VLANs). VLANs partition nodes in the network into communities of interest. Servers within one community can only communicate with servers that belong to the same community (or VLAN). Such practice allows both performance isolation and network scalability, since packets exchanged

within a VLAN do not stretch to the entire network. Additionally, to exploit path redundancy in the network, the Multiple Spanning Tree Protocol (MSTP), an extension of STP, allows different VLANs to use different spanning trees traversing diverse physical links and switches. A traffic engineering framework for MSTP in large data centers is presented in [24] where the authors focused on the mapping of each VLAN into a spanning tree to improve the overall network utilization. Smart Path Assignment In Networks (SPAIN) is presented in [18] to provide multipath forwarding by exploiting commodity off the shelf Ethernet switches; SPAIN exploits the redundancy in an arbitrary network topology and pre-computes paths that are merged into trees which later are mapped into VLANs to achieve higher bisection throughput. Multipath routing in data center networks is also addressed through ensemble routing [21]; the framework aggregates individual flows into routing classes which are then mapped into VLANs to load balance the traffic across the network links. Traffic engineering in data centers with ensemble routing is studied in [20] where the authors leverage the multi-path routing to enable traffic load balancing and quality of service provisioning. The authors noted that a key challenge for intra-data center traffic engineering is the optimal flow mapping into VLANs (known as the VLAN assignment problem) to achieve load balancing. The VLAN assignment problem is NP-complete with a large search space [19]. For instance, for a network with v VLANs and c flows, there are $|v|^c$ possible mappings [20]. Finding the best mapping is therefore a complex combinatorial problem. Several prior works have attempted to address this or simpler variation of this problem. For instance, SPAIN [18] proposed a greedy heuristic for packing flows or paths into a minimal set of VLANs and the subgraphs containing these VLANs are dynamically constructed. A constraint based local search based on Constraint Programming (CP) is presented in [24] for mapping flows into VLANs, assuming the set of VLANs is given. The authors of [20] used Markov Approximation techniques to solve the mapping problem and designed approximation algorithms with close to optimal performance guarantees. The authors assume the VLANs or the spanning trees are pre-constructed and their method assigns routing classes to these VLANs with the objective of minimizing link congestion.

In this chapter, we consider the problem of traffic engineering in data center networks; namely, we address the problem of mapping traffic flows into VLANs; similar to [20], a separate spanning tree is constructed per each VLAN. As the number of spanning trees in a network could be very large (e.g., in a fully connected graph with n nodes, there are as many as n^{n-2} spanning trees), selecting the most promising spanning trees to map the traffic flows onto is a very challenging and complex combinatorial problem. This dissertation

jointly addresses the problem of tree construction and flow mapping, a seemingly very large combinatorial problem. To keep track of the problem, we follow a primal-dual decomposition approach using Column Generation [62]; here, the problem is divided into two subproblems: a master and a pricing. The latter builds sub mappings (a single spanning tree with mapped flows), and the former selects among the sub mappings, the global mapping of the problem. Our method proves to be highly scalable in addressing the joint VLAN mapping and tree construction problem.

2.2 Related Work

Today's data center networks are expected to provide high network utilization to the large number of distinct services they support. These networks are also expected to provide guaranteed performance for the various hosted tenants sharing the same networking infrastructure. Therefore, Intelligent traffic engineering mechanisms inside a data center become therefore of utmost importance to achieve better load balancing and guarantees on the quality of service. Recently, the problem of traffic management in data center networks has been receiving numerous attention with the goal of providing predictable performance. The issue of performance predictability becomes of particular interest given the nature of data center topologies, which tend to be oversubscribed [17].

Hedera [63] is a dynamic flow scheduling system for data center networks based on multistage switch topologies. Hedera depends on a central scheduler that measures link utilization in the network and moves flows from highly utilized links to less utilized ones. When the scheduler detects a flow with augmenting bandwidth demand that exceeds a certain threshold, it computes a non-conflicting path to route this flow in order to improve the bandwidth-bisection in the network. Hedera uses simulated annealing and the global first fit heuristics for path computation. When compared to the Equal Cost Multiple Paths (ECMP) protocol, Hedera was found to yield substantial throughput performance gains. MicroTE [22] is a fine grained traffic engineering approach that consists of exploiting the short-term predictability in data center traffic. A central controller is used to collect and analyze traffic in data centers for traces of similarities. Then, traffic is segregated between predictable and unpredictable, where predictable traffic demands are routed optimally using a Linear Programming (LP) model, while the remaining traffic is routed using the weighted ECMP protocol. However, it is noted in [64] that such centralized solutions face scalability and fault tolerance challenges. Accordingly, the authors of [64] proposed a method for load-aware flow routing in

data center networks which does not require centralized control. Their approach achieves a maximal multi-commodity flow while tolerating failures. The multi-commodity flow problem is decomposed into two subproblems, a slave dual which can be solved at the end hosts and a master dual solved locally at each switch.

Now to achieve high bisection bandwidth in data center networks, the authors of [18] proposed SPAIN, which provides multipath forwarding using Layer-2 Ethernet switches over arbitrary topologies. Here, SPAIN aims at finding multiple paths between every pair of nodes, and then grouping these paths into a set of trees, each tree is mapped as a separate VLAN onto the physical network. The path assignment into VLANs uses a greedy packing heuristic with the objective of using the minimum number of VLANs. SPAIN's performance was validated through both simulations and experiments and has shown to achieve superior goodput over STP. Traffic engineering in data center networks has been a topic of surging interests recently; particularly, exploiting path redundancy in the network and mapping paths into VLANs to achieve higher utilization and better load balancing is addressed in [24]. The authors leverage the MSTP protocol in large data centers and presented a heuristic, based on local search algorithm, to select good spanning trees to map the traffic demand matrices while achieving minimum overall link utilization. Similar to [24], the authors of [20], [65] have addressed the problem of traffic engineering in data center network by managing aggregate (ensemble) flows rather than individual flows. Essentially, the problem considered is that of mapping routing classes to VLANs to achieve load balancing. In [65], a VLAN placement algorithm is presented for growing spanning trees to reach all switches in the network and subsequently traffic splitting onto those VLANs is presented (through linear programming) to achieve load balancing. However, the heuristic construction of spanning trees does not provide any guarantees on the quality of the generated solution. More recently, the authors of [20] considered the problem of traffic engineering with ensemble routing and noted the combinatorial challenge of optimizing the assignment of routing classes into VLANs. They used the Markov approximation framework for approaching the mapping problem and developed approximation algorithms with close to optimal performance.

2.3 The VLAN Assignment Problem

In this section, we present a formal definition of the VLAN assignment problem, and we also show through an illustrative example the exponential search space that the VLAN assignment problem poses.

2.3.1 Problem Definition

We consider a data center network whose underlying physical topology is represented by a graph $G^s = (N, L)$, N is the set of substrate nodes and L is the set of physical links. Each link connects a pair of substrate nodes (i, j) and has a capacity $b_{i,j}$. We assume a set of incoming flows, and similar to [20], we focus on the problem of mapping traffic flows into VLANs. We assume each flow has a bandwidth demand of δ_c and let C be the set of all flows and V be the set of VLANs. We acknowledge that the maximum number of VLANs a switched Ethernet network supports is 4096, however, the number of spanning trees in the network could substantially exceed that. Given the traffic matrix, finding the most useful spanning trees to these VLANs is a combinatorially complex and challenging problem. Thus, the VLAN assignment problem can be formulated as follows:

Problem Definition 2.1. *Given a network $G^s = (N, L)$, the set of VLANs V, and a set of flows C, each with bandwidth requirement of δ_c; find the optimal mapping of flows to VLANs that balances the load across the network.*

In this work, we attempt to balance the load across the network to minimize congestion, therefore minimizing the maximal link load becomes our objective. Solving the VLAN assignment problem under this objective has been shown [20] to be NP-Complete via a reduction from the minimum-weight Steiner tree problem. In addition, the VLAN assignment problem also suffers from an exponential search space, both in the number of VLANs present in a given network, as well as in the number of different possibilities in which connection demands can be mapped to the VLANs. These two elements render the problem heavy, for the following reasons:

- First, enumerating all spanning trees in a given network can be problematic and tedious, particularly as there are no efficient algorithms to perform this enumeration, not to mention the large number of spanning trees present in a medium-sized network. For instance, for a clique topology, there are $n^{(n-2)}$ spanning trees, by the well-known Cayley's formula [19].

- Second, since the Ethernet standards can only support 4096 VLANs, a large number of the enumerated trees will never be used, and the challenge becomes to find a subset that can lead to optimal traffic split.

- Third, the number of different possibilities in which flows (or paths) can be mapped to available VLANs is exponential. This is similar to the various ways n distinct objects

can be distributed into m different bins with k_1 objects in the first bin, k_2 in the second, etc. and $k_1 + k_2 + ...k_m = n$. This indeed is obtained by applying the multinomial theorem where $\sum_{k_1+k_2+...+k_m=n} \binom{n}{k_1+k_2+...+k_m} = m^n$. Therefore, for C flows and V VLANs, there are $|V|^C$ different mapping possibilities.

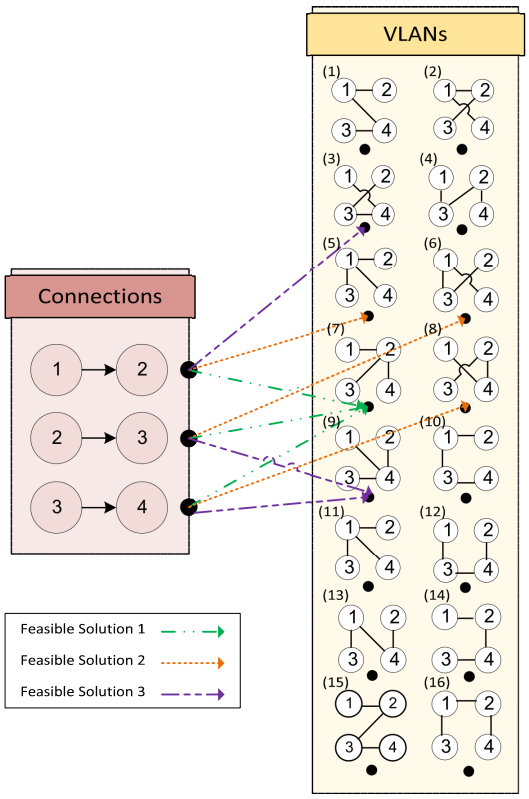


Figure 2.1: Traffic Flows to VLAN mapping

To further convey the complexity of the problem, we consider 3 traffic flows in a complete graph K_4; in such a 4-nodes topology, there are 16 different spanning trees. Mapping the traffic flows to these spanning trees involves a large number of possibilities as shown in Figure 2.1. For instance, one possible solution would be to assign all demands to a single VLAN. Another feasible solution would be to map each flow to a different VLAN. Or, map two flows to the same VLAN, and place the remaining one in a different VLAN. In fact, with 16 different spanning trees to choose from, there are 16^3 (4096) different mapping possibilities. This number gets more dramatic as the network grows in size. For instance, in a K_5 complete graph with 3 connections, the number of mapping possibilities is approximately 2 millions! Clearly, while exhaustive enumeration of the spanning trees as well as the mappings will yield to the optimal solution, such enumeration is prohibitively computationally expensive. We explore in a subsequent section the possibility of only generating a small subset of

such configurations or mappings. Our approach uses a column generation framework where only configurations/mappings that are deemed good are constructed through a reduced cost function (more details are presented in the sequel).

2.3.2 Problem Formulation

In this formulation, we assume the set of VLANs is predetermined (through offline enumeration). Given a traffic demand matrix, we aim at balancing the traffic load across the network through minimizing the maximum link utilization.

Below we present the mathematical model of the VLAN assignment problem:

- Parameters:

$G^s = (N, L)$: the physical network with N nodes and L links.

b_{ij}: capacity of link $(i, j) \in L$.

C: set of flows, each with an origin $o(c)$, a destination $d(c)$, and a bandwidth demand of δ_c.

V: set of VLANs.

$$\sigma_{ij}^{v} = \begin{cases} 1, & \text{if VLAN } v \text{ traverses link } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

- Decision Variables:

$$x_c^v = \begin{cases} 1, & \text{if flow } c \text{ is mapped to VLAN } v, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ij}^{c,v} = \begin{cases} 1, \text{if flow } c \text{ is mapped on } v \text{ and routed through link } (i,j), \\ 0, \text{otherwise.} \end{cases}$$

t_{ij}^v: total traffic contributed by VLAN v on link (i, j).

u_{ij}: utilization of link (i, j).

The VLAN assignment problem is formulated next:

$$Min\ Max_{(i,j)\in L}\ \ u_{ij}$$

Subject to

$$\sum_{j:(i,j)\in L} y_{ij}^{c,v} - \sum_{j:(j,i)\in L} y_{ji}^{c,v} \leq 1 \quad \forall c \in C, i = o(c) \tag{2.1}$$

$$\sum_{j:(i,j)\in L} y_{ij}^{c,v} - \sum_{j:(j,i)\in L} y_{ji}^{c,v} = 0 \quad \forall c \in C, i \in N \backslash \{o(c), d(c)\} \tag{2.2}$$

$$\sum_{j:(i,j)\in L} y_{ij}^{c,v} - \sum_{j:(j,i)\in L} y_{ji}^{c,v} \geq -1 \quad \forall c \in C, i = d(c) \tag{2.3}$$

$$\sum_{v\in V} x_c^v \leq 1 \quad \forall c \in C \tag{2.4}$$

$$y_{ij}^{c,v} \leq x_c^v \quad \forall v \in V, c \in C, (i,j) \in L \tag{2.5}$$

$$t_{ij}^v = \sum_{c\in C} y_{ij}^{c,v}.\delta_c \quad \forall v \in V, (i,j) \in L \tag{2.6}$$

$$\sum_{v\in V} t_{ij}^v \leq b_{ij} \quad \forall (i,j) \in L \tag{2.7}$$

$$y_{ij}^{c,v} \leq \sigma_{ij}^v \quad \forall v \in V, c \in C, (i,j) \in L \tag{2.8}$$

$$u_{ij} = \frac{\sum_{v\in V} t_{ij}^v}{b_{ij}} \quad \forall v \in V, (i,j) \in L \tag{2.9}$$

Constraints (2.1-2.3) are the flow conservation constraints. Constraint (2.4) indicates that one flow can be mapped on at most a single VLAN only. Constraint (2.5) indicates that every flow will only be routed through the links that form the VLAN onto which that former is mapped. This constraint allows to establish a strict connection between where a given flow is routed, and how. Constraint (2.6) calculates the amount of traffic contributed by each VLAN v on a link (i, j). Constraint (2.7) ensures that the amount of traffic routed on a given link, does not exceed the capacity of the link. Constraint (2.8) ensures that a connection cannot use a link in a VLAN, if that link does not belong to the spanning tree of that VLAN. Finally, Constraint (2.9) calculates the utilization of every link in the network.

2.4 Decomposition Model

In the following section, we present our column generation model, as well as, the intuition by which we decomposed the original VLAN assignment problem into two subproblems towards

a scalable solution to the aforementioned hurdle.

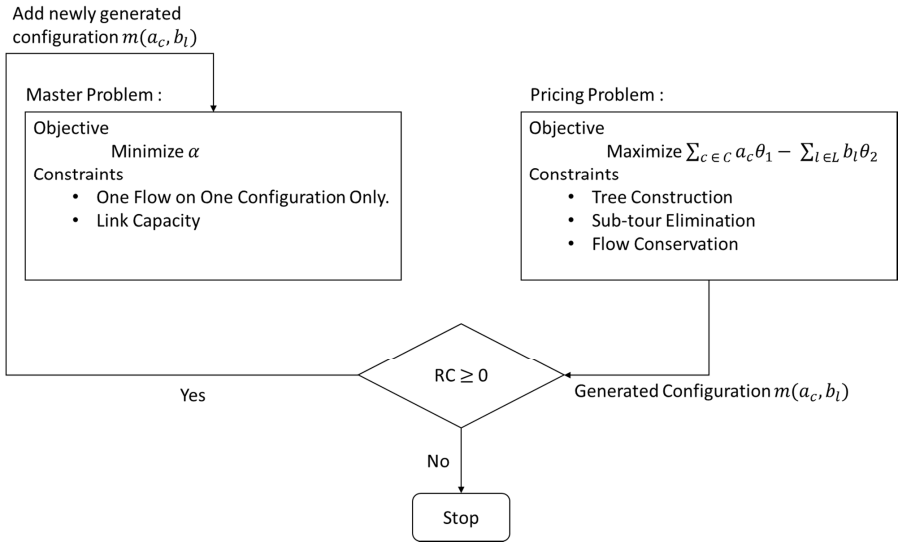


Figure 2.2: Flow Chart of CG1 Model

2.4.1 Column Generation

Column Generation is an efficient method for solving large LP problems [62]. Normally, given an LP, the algorithm begins with an initial subset of configurations (columns) ($\mathcal{M}_0$) that satisfies all the constraints. At each iteration, a new configuration ($m \in \mathcal{M}$, where $\mathcal{M}$ is the set of all possible configurations or columns), that ameliorates the objective function, is added to the initial set ($\mathcal{M}_0 = \mathcal{M}_0 \cup m$). Thus, unlike the Simplex method, column generation only explores a subset of the variables, instead of enumerating all of them. In every iteration of the Simplex method, the goal is to find the next non-basic variable to enter the set (basis), which is the one with the minimum reduced cost coefficient. The Simplex method resorts to calculating the reduced cost coefficient of all non-basic variables whereas column generation alleviates this by only finding this next-entering variable. Column generation decomposes the initial problem into two sub-problems, a master (LP model) and a pricing (ILP). The master is in charge of determining if the explored configurations satisfy all the constraints. The pricing model is in charge of finding a new configuration to be passed on to the master. If the newly found configuration was deemed feasible (and will improve the objective) by the master, it will be added as a new column, hence the name of the method. The pricing's objective function is in fact the reduced cost coefficient of the master. Thus, the master model can be referred to as the primal, while the pricing as the dual. The master and the

pricing problems alternate until no new configurations are found with a negative reduced cost coefficient (in the case of a minimization problem) which indicates that optimality is reached.

2.4.2 The Intuition Behind our Decomposition Approach

Decomposing the VLAN assignment problem to be solved using the column generation method is not straightforward. To achieve a concise and vigorous decomposition, we attempted to reformulate the problem in a way that resembles the famous cutting-stock problem [62]. In the cutting stock problem, the difficulty resides in the very large number of cuts available to choose from. This indeed resembles the VLAN assignment problem where a very large number of mapping possibilities between flows and VLANs exists.
Similar to the cutting stock problem, we envision a cut in the VLAN assignment problem as a unique mapping of a subset of flows to a particular VLAN. Finding the optimal set of cuts is nothing but finding the optimal way various flows can be mapped to a subset of the available VLANs. Following the above, we can divide our problem into two subproblems: the pricing will be in charge of finding a new configuration, while the master will ensure that these configurations do not violate the capacity constraints. In the following section, we will present our decomposition model.

2.4.3 The Column Generation Model for the VLAN Assignment Problem (CG1)

We will now introduce our column generation model for the VLAN assignment problem. We have modified the problem definition to include not only finding the optimal mapping of flows to VLANs, but also finding the optimal set of VLANs (spanning trees) onto which these flows will be mapped. In fact, as the number of possible spanning trees in a network can be very large, limiting the search space to those VLANs that offer a good quality solution will greatly improve the runtime of the model. We introduce a new variable m that denotes a configuration. A configuration is defined as a VLAN v onto which flows are mapped.
Thus, the problem definition of the column generation model can now be stated as follows:

Problem Definition 2.2. *Given a network $G^s = (N, L)$, and a set of flows C, each with bandwidth demand δ_c, find the optimal set of configurations, such that the maximum link utilization is minimized.*

At every iteration, the pricing model will generate a new configuration that enhances the objective function. These generated configurations will be sent to the master to ensure that they do not violate the link capacity constraints, as well as the constraint of mapping a connection onto only one VLAN. To generate a new configuration, the pricing requires building a spanning tree and mapping flows onto the generated trees. Each new configuration has to improve the master's objective function and the reduced cost coefficient. The pricing will keep running until its objective function becomes greater than or equal to 0 (in the case of a minimization problem) which indicates that optimality is reached. The master problem can be formulated as follows:

The Master Problem:

- Parameters:

$G^s = (N, L)$: the network with N nodes and L links.
C: set of flows, each with an bandwidth demand δ_c.
b_l: capacity of link $l \in L$.
$\mathcal{M}$: set of *all* configurations, each indexed by m.
$\mathcal{M}_0$: set of explored configurations.
$a_m^c = \begin{cases} 1, & \text{if flow } c \text{ is mapped to configuration } m, \\ 0, & \text{otherwise.} \end{cases}$
t_m^l represents the traffic flow contributed by configuration m on link l.

- Decision Variables:

$z_m = \begin{cases} 1, & \text{if configuration } m \text{ is selected,} \\ 0, & \text{otherwise.} \end{cases}$
α: represents the maximum link utilization.

$$Minimize \;\; \alpha$$

Subject to

$$\sum_{m \in \mathcal{M}_0} a_m^c z_m = 1 \quad \forall c \in C \tag{2.10}$$

$$\frac{\sum_{m \in \mathcal{M}_0} t_m^l z_m}{b_l} \leq \alpha \quad \forall l \in L \tag{2.11}$$

$$z_m \in \{0,1\} \quad \forall m \in \mathcal{M}_0 \tag{2.12}$$

$$\alpha \leq 1 \tag{2.13}$$

The objective of the master problem is to achieve load balancing by minimizing the maximum link utilization. However, we reformulate the min-max constraint by defining a new variable α that represents the highest link utilization, and we aim to find the optimal value α such that none of the links in the network have a utilization that exceeds this value. Constraint (2.10) ensures that each flow, if mapped, can only be running on one configuration or one VLAN. Constraint (2.11) ensures that the physical capacity of the links in the network is not violated. Constraint (2.12) is the integrality condition that indicates if a configuration m is selected or not. However, for solving the master, we relax this constraint and allow z_m to take any value in the range between 0 and 1. This becomes an LP relaxation of the original problem and when solved yields a lower bound solution to the original problem. Finally, constraint (2.13) ensures that α could be at most equal 1, which indicates that at least one link in the network is fully saturated.

The Pricing Problem:

Note that a_m^c and t_m^l are both parameters in the master problem which are obtained after solving the pricing subproblem. During every iteration, when the master problem is solved, we need to verify the optimality of the solution. If it is optimal we conclude our search, or else the pricing attempts to find a new column to join the current basis which may improve the master's objective. This can be achieved by examining whether any new configuration which has not been added to the basis has a negative reduced cost. We denote the dual variables corresponding to (2.10) and (2.11) by Θ_1 and Θ_2 respectively. The Reduced Cost (RC) for an off basis column is expressed as:

$$RC = \sum_{c \in C} a_c \Theta_1 - \sum_{l \in L} t_l \Theta_2 \tag{2.14}$$

When the master's objective is a minimization function, the standard pivoting rule of the simplex method is to choose a new column (configuration) such that $\sum_{c=1}^{C} a_c \Theta_1 - \sum_{l=1}^{L} b_l \Theta_2$ is maximum; the column (configuration) that is found is added to the basis of the master model. The master model is solved, again, with the new basis to obtain a new solution, and the dual variable is passed to the pricing which is again solved. The two subproblems are solved iteratively until there is no off-basis column with a positive reduced cost found and therefore the solution is optimal. Indeed, this requires that the last Simplex iteration of the pricing model be solved to optimality to ensure that there is no off-basis column with positive

reduced cost remains unexplored. Figure 2.2 presents an illustration of how the master and the pricing models work iteratively and jointly until the optimal solution is found (RC $\leq$ 0). Before we present the pricing model, we introduce a new set of decision variables for the pricing problem that are needed for spanning tree construction. These decision variables are listed below:

- Decision Variables:

$$r^i = \begin{cases} 1, & \text{if node } i \text{ is the root node of the tree } T, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_i^j = \begin{cases} 1, & \text{if node } j \text{ is the parent node of } i \text{ in } T, \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{ij}^c = \begin{cases} 1, & \text{if flow } c \text{ is routed on link } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

$$a_c = \begin{cases} 1, & \text{if flow } c \text{ is routed,} \\ 0, & \text{otherwise.} \end{cases}$$

V_i : set of adjacent nodes for node $i \in N$.

t_l represents the traffic flow on link l.

Thus, the mathematical model of the pricing problem can be stated as follows:

$$Maximize \quad RC$$

Subject to

$$\sum_{i \in N} r^i = 1 \tag{2.15}$$

$$\sum_{j \in V_i} y_i^j + r^i = 1 \quad \forall i \in N \tag{2.16}$$

$$y_i^j + y_j^i \leq 1 \quad \forall (i,j) \in L \tag{2.17}$$

$$\sum_{i,j \in V} y_i^j + y_j^i \leq |V| - 1 \quad V \sqsubset N, i, j \in V \tag{2.18}$$

$$\sum_{j:(i,j)\in L} x_{ij}^c - \sum_{j:(j,i)\in L} x_{ji}^c \leq 1 \quad \forall c \in C, i = o(c) \tag{2.19}$$

$$\sum_{j:(i,j)\in L} x_{ij}^c - \sum_{j:(j,i)\in L} x_{ji}^c = 0 \quad \forall c \in C, i \in N \backslash \{o(c), d(c)\} \tag{2.20}$$

$$\sum_{j:(i,j)\in L} x_{ij}^c - \sum_{j:(j,i)\in L} x_{ji}^c \geq -1 \quad \forall c \in C, i = d(c) \tag{2.21}$$

$$\sum_{c\in C} x_{ij}^c \leq 1 \quad \forall (i,j) \in L \tag{2.22}$$

$$t_l = \sum_{c\in C} x_{ij}^c \delta_c \quad \forall (i,j) \in L, s_l = i, d_l = j \tag{2.23}$$

$$x_{ij}^c \leq y_i^j + y_j^i \quad \forall (i,j) \in L, c \in C \tag{2.24}$$

$$a_c = x_{ij}^c \quad \forall c \in C, i = o(c) \tag{2.25}$$

Constraint (2.15) ensures that a tree has a single root. Constraints (2.16-2.18) model the spanning tree construction. Constraint (2.16) ensures that each node, except the root node, will have a parent node. Constraint (2.17) prevents cycling parent-son relationship. Constraint (2.18) represents the sub-tour elimination constraint. Constraints (2.19-2.21) are the flow conservation constraints. Constraint (2.22) ensures that a link can only accommodate a single flow in this particular configuration. This constraint, on one hand, can limit the number of generated configurations that violates the link capacity constraint in the master, and on the other hand, allows us to further decompose the pricing problem at each iteration. Constraint (2.23) calculates the amount of traffic routed on each link l. Constraint (2.24) ensures that a link cannot be used if it does not belong to the spanning tree. Finally, Constraint (2.25) indicates whether a flow c is routed or not.

Note that, once the relaxed (fractional) VLAN mapping problem is solved, as mentioned earlier, the obtained solution is a lower bound to the optimal solution of the original problem. To obtain the ILP solution, we solve the master program one last time with z_m assuming integer values and this allows us to obtain a solution to the integral mapping problem. We acknowledge however that the obtained solution is only an approximation of the optimal one and better quality solutions may be obtained by employing branch and bound methods.

2.4.4 The modified Column Generation Model (CG2)

While the CG decomposition substantially overcomes the computational complexity of the original mapping problem by avoiding the complete enumeration of the flows-to-VLAN mappings, it is to be noted that the pricing still exhibits some scalability issues given its ILP nature. In this section, we try to further enhance the running time of the proposed CG. Namely, we modify our pricing model by relaxing the tree construction constraints. In fact, finding a tree requires the model to first choose a single root among the available nodes. Upon the selection of a root, each node has to select one of its eligible neighbours as a parent. With a choice of V adjacent nodes for each node out of N, this results in $|N|^V$ possible combinations. As the size of the network grows, the number of tree construction constraints increases substantially. Instead of searching for a new tree at every master-pricing iteration, we opt to enumerate spanning trees offline (using Dijkstra's algorithm) and let the pricing select suitable ones which improve the value of the objective of the master's subproblem, until no further improvements (no spanning tree will yield positive reduced cost) are obtained. In the worst case, the pricing will navigate through the whole set of pre-determined spanning trees. To guarantee a well diversified set, every time the algorithm returns a tree, we increase the weight of the links in the generated tree. This ensures that our algorithm will try to avoid these links during the next tree construction.

The *SpanningTreeGeneration* function is illustrated in Algorithm 2.1. The algorithm takes as input the number of the spanning trees that need to be generated, denoted by k, and a constant number denoted by *increment*, that represents the value to be used to increment the link weights at the end of every tree construction iteration. This constant is set to a large value in order to ensure that a link will only be included in multiple spanning trees if there are no other detours. At the end, the algorithm returns the set all spanning trees found, denoted by ST. The algorithm begins by setting the link weights of all edges in the network to 1. Next, the algorithm will loop k times until k spanning trees are constructed. If at a given iteration, a redundant spanning tree st is found (meaning it already belongs to the set ST), the algorithm terminates, as it can no longer find unique spanning trees. At the beginning of every tree construction iteration, the algorithm will pick a random source from the set of nodes in the network to become the root node (denoted by s). Next, the model will run the Dijkstra algorithm to find the shortest path from the root node s to all other nodes in the network. At every iteration, the set of shortest paths are aggregated to form a spanning tree st. This guarantees that the tree will not contain any cycles. If the constructed tree is unique, the algorithm will increment the weight of every edge in that

Algorithm 2.1 GenerateSpanningTrees Algorithm

```
1: Given:
2: G^s = (N, L) : an arbitrary topology
3: k : number of Spanning Trees to be generated
4: increment : edge weights increment value
5:
6: ST = { };
7:
8: for (l ∈ L) do /* Initialize the weight of all edges*/
9:     w(l) = 1;
10: end for
11:
12: while (|ST| ≤ k) do
13:     st = { };
14:
15:     /* Select a random node to become the root node*/
16:     s = Random(N);
17:     N' = N - { s };
18:
19:     for (d ∈ N') do
20:         p = Dijkstra(s, d, L);
21:         st = st ∪ p;
22:     end for
23:
24:     if (ST.contains(st)) then
25:         Break;
26:     end if
27:
28:     if (containsCycle(st)) then
29:         continue;
30:     else
31:         for (l ∈ st) do
32:             w(l) += increment;
33:         end for
34:         ST = ST ∪ st;
35:     end if
36:
37: end while;
38: return ST;
```

tree, and then will add that tree to the set ST.

Algorithm 2.2 Modified Column Generation Approach

```
1: Given G^s=(N, L) /*an arbitrary topology*/
2: ST = GenerateSpanningTrees;
3: RC = -K /*K is a very large positive number*/
4: M_0 = { };
5: while (!Terminate) do
6:     st = ST.next();
7:     /*Initialize Set of Configurations for spanning tree st*/
8:     M_st = { };
9:     while ((RC ≤ 0)) do
10:        RC = Run Master;
11:        /*m is a new Configuration*/
12:        m = Run Pricing(RC,st);
13:        RC = m.RC;
14:        M_st = M_st ∪ m;
15:    end while
16:    if (M_st.length ≤ 1) then
17:        Terminate = TRUE;
18:    end if
19:    M_0 =M_0 ∪ M_st;
20: end while
```

Algorithm 2.2 illustrates the methodology of our heuristic model: Given the topology of the substrate network, represented by $G^s = (N, L)$, the model starts by calling the $GenerateSpanningTree$ function to enumerate multiple spanning trees that will be stored in a dedicated set ST. As we have previously mentioned, the $GenerateSpanningTree$ function adopts the Dijkstra algorithm [66] with link weights increment. Next, at the beginning of every iteration of the pricing model, we provide the pricing with a new spanning tree st from the set ST, and for every tree, we initialize a set M_{st} that represents the list of configurations for that given tree. The master and the pricing iterate over the same given tree, as long as it produces feasible configurations (a configuration with positive reduced cost). At the end of every iteration, the explored configuration m is added to the set M_{st}. Once the tree becomes non-bearing (no feasible configurations can be found), a new tree from the initial set is selected. This procedure persists until one tree returns an infeasible configuration in the first trial, meaning that its M_{st} remains empty at the end of the first master/pricing iteration, in that case, the program terminates. Our numerical results have shown that our heuristic model achieves a great improvement in runtime and scalability over our initial exact decomposition model without sacrificing the quality of the final solution.

2.5 Performance Evaluation

In this section, we numerically evaluate the proposed VLAN mapping methods. In particular, we present comparisons between the ILP and its two decomposition variations. We refer to the column generation models presented in sections 2.4.3 and 2.4.4 as CG1 and CG2. The objective of our comparisons is to study the effectiveness of the designed methods in terms of quality of the obtained solutions and runtime. We also present comparisons with state of the art traffic engineering methods in data centre networks; namely, we compare the performance of our proposed decomposition methods against the STP, SPAIN, and ECMP. The purpose of this study is to see how well these protocols compare to the optimal obtained results. All our numerical evaluations are conducted using CPLEX version 12.4 on a pentium IV machine at 3.4 GHz with 8 GB RAM.

2.5.1 Numerical Results

Table 2.1: Runtime and Optimality Gap Comparative Analysis (500 Flows)

Topology	ILP Model		CG 1			CG 2		
	#Config	Runtime(s)	#Config	Runtime(s)	Gap	#Config	Runtime(s)	Gap
K_3	3^{500}	5	452	230	1%	548	120	0%
K_4	16^{500}	24	579	318	2%	760	266	2%
K_5	125^{500}	452	660	654	2%	866	455	3%
K_6	1296^{500}	> 81823	1111	1020	2%	1322	546	4%

We start by evaluating how our model performs when compared to the ILP model of section 2.3.2. We consider a clique topology as it allows to determine the number of all possible spanning trees and configurations, which would provide a solid benchmark for comparison. We suppose each node in the clique represents a switch, and that each switch is connected to one host with an immediate link. We consider a traffic demand matrix of 500 sessions/flows, each pair of communicating nodes is randomly chosen and each demand is of one unit (normalized demand). As our focus is to map flows onto VLANs while balancing the traffic across the network (traffic engineering problem), we assume links with enough capacity to route all the demands. Table 2.1 summarizes our findings. Here, we present the average of 10 executions for each topology. The results show that the number of configurations explored by our decomposition models are substantially smaller when compared to the search space of the ILP model. In fact, for a K_3, the ILP model navigates through 3^{500} configurations, while CG1 and CG2 explore 452 and 548 configurations respectively (observe

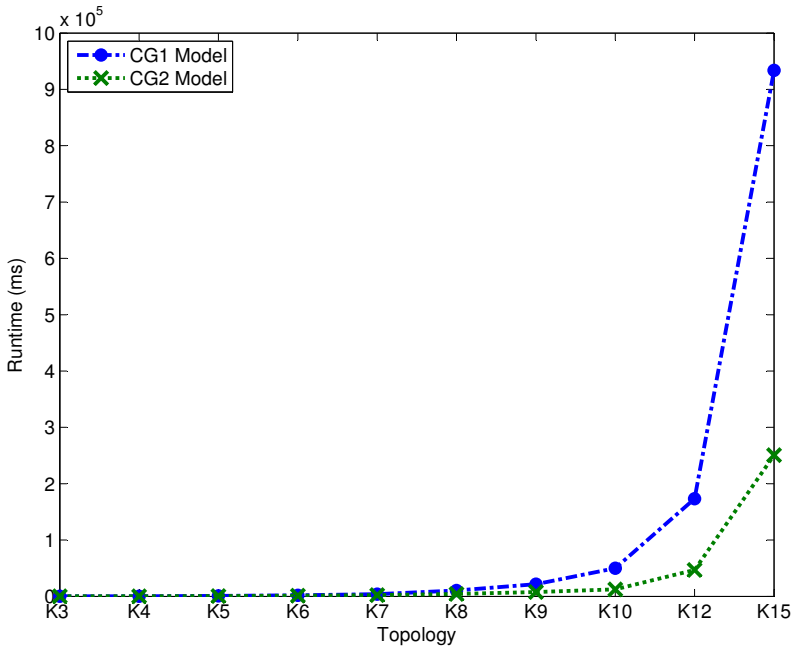


Figure 2.3: Towards Scalable Traffic Management - Runtime Analysis of CG1 and CG2

that the fraction of explored configurations by our decomposition models is less than 1% of the search space of the pure ILP model and this fraction becomes negligible as we experiment with larger clique topologies). Next, we look at the runtime of the models, and we notice that, as opposed to CG1 and CG2, the runtime of the ILP increases exponentially as the size of the network increases. This is attributed to the fact that both CG1 and CG2 examine much smaller number of configurations (only good configurations are enumerated by the pricing subproblem) for solving the mapping problem. It is interesting to note that for smaller size networks, the ILP exhibits faster runtime. Indeed, this is due to the fact that with the ILP, the trees are always enumerated offline and the mathematical model solves the mapping problem. However, both with CG1 and CG2, the trees are explored dynamically as we seek to improve the quality of the solution. Notice that for a K_6 network, the ILP failed to obtain a solution after running the model for 22 hours. We also observe that CG2 consistently outperforms CG1 in terms of runtime; this is due to the fact that the pricing model of CG2 is much smaller than that of CG1. Indeed, as mentioned in section 2.4.4, the tree construction in CG2 is not done within the pricing model (as in CG1), but rather, trees are enumerated offline and explored dynamically as we search for best configurations. The size of the ILP pricing of CG1 is much bigger and thus less scalable in terms of runtime. Figure 2.3 compares the runtime of both CG1 and CG2 over larger clique topologies of 3 to 15 nodes with all to all connection demands. The figure shows that the runtime of CG2 is

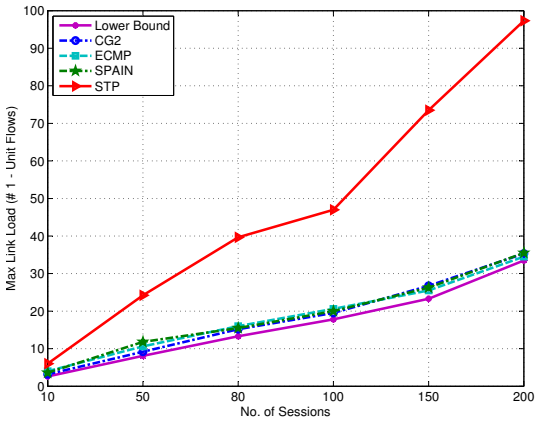
much faster than CG1, in fact as we move towards cliques with more than 8 nodes, CG2 becomes at least three times faster.

Finally, we examine the quality of the obtained solutions. We show, in Table 2.1, the % gap of the solution obtained by CG2 and CG1 to the one obtained by the ILP. In most cases, the gap is less than 2% for CG1 and 4% for CG2. The gap between the solution of CG1 and that of the ILP is attributed to the manner through which we obtained the ILP solution for CG1. Namely, as we have solved the fractional mapping problem (lower bound), we resorted to a straightforward approach for solving its ILP version. We believe this gap may be reduced if we employ a more effective branch and bound method for obtaining the integral solution. Now the gap of CG2 is attributed to the heuristic nature of the methodology followed for solving the mapping problem. That is, the more spanning trees are explored, the better the quality of the solution gets, but that comes at the cost of increased runtime of the model.

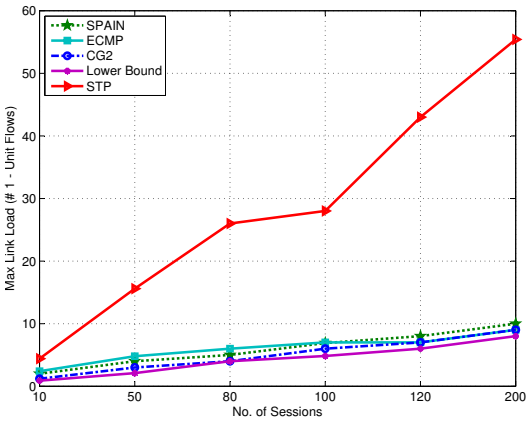
2.5.2 Comparative Analysis

In this section, we study the performance of different traffic engineering schemes; namely SPAIN [18], STP, and ECMP [67], against the obtained solutions from our design methodology. We used CG2 (owed to its better scalability) for obtaining benchmark solutions. We use STP as a base for comparison, since it is widely used in Ethernet networks. Our objective is to study how well the traffic is balanced in the network and to measure the overall network goodput. For each experiment, We executed multiple test cases for different number of sessions in the range [10,50,80,100,150,200]. In each session, we consider random traffic demands between random pair of hosts. In all test cases, the results are averaged over multiple runs.

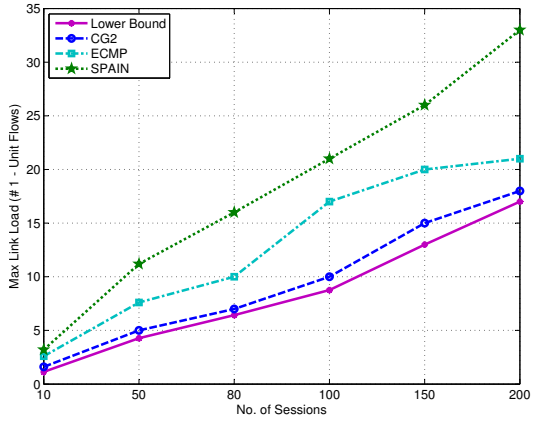
2.5.2.1 Load Balancing



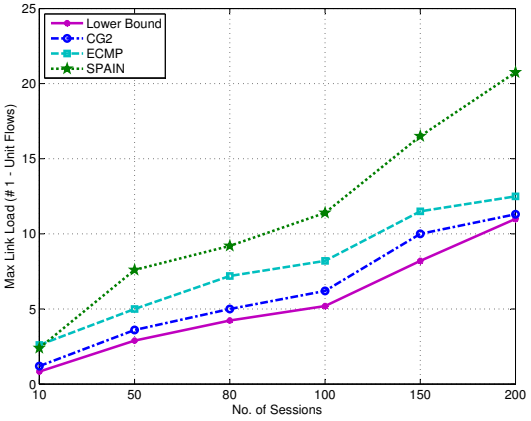
(a) FatTree k=4



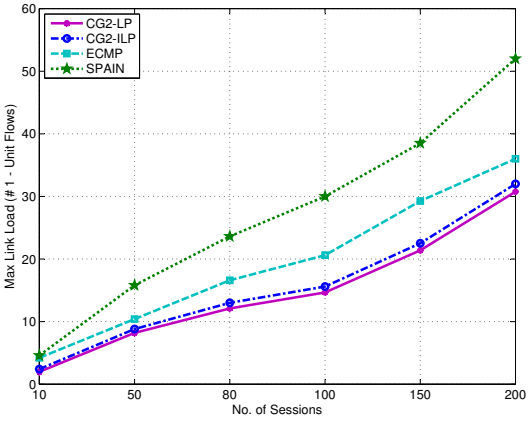
(b) FatTree k=8



(c) Clique K_8



(d) Random Topology R1 - 14 switches, 19 links



(e) Random Topology R2 - 11 switches, 25 links

Figure 2.4: Max Link Load Comparison between CG2, ECMP, SPAIN, and STP

We begin by studying the load balancing capability of our model. For this purpose, we use the FatTree network, since it is a well-adopted topology for cloud data centers [17]. A FatTree network consists of 3 layers of interconnected ethernet switches. The links connecting the switches are of 10 Gbps, while those connecting the server racks to the second layer aggregate switches are of 1 Gbps, hence the name of this topology.

We compare the performance of the legacy STP protocol, ECMP and SPAIN, against results obtained from the CG2 model using a FatTree network (k=4 and k=8); the results are illustrated in Figures 4(a) and 4(b) respectively. It is clear that the STP protocol has inferior performance, achieving very high link load. This is indeed expected, since STP concentrates the load on a particular set of links in the network and does not benefit from the existence of multipaths, thus quickly congesting the selected links. On the other hand, both ECMP and SPAIN spread equally the load across the network, achieving very close performance to those results obtained by CG2.

Next, we further study the quality of the solutions obtained by ECMP and SPAIN and compare their solutions to those obtained by CG2-LP (lower bound) and CG2-ILP (CG2) (as shown in Figures 4(a) and 4(b)). We observe that for a FatTree network (k=4, k=8), both ECMP and SPAIN perform quite well; namely, we see that the CG2 solution has a gap of 11% to the lower bound while ECMP and SPAIN both have a gap of 16%. For a FatTree with k=8 (Figure 4(b)), the CG2 model provides an overall 16% gap from lower bound, while ECMP and SPAIN both show a 30% gap. It is to be noted that as the number of sessions increases, the gap decreases. For instance, for FatTree with k=4 and 200 sessions (Figure 4(a)), ECMP's gap to the lower bound reaches 2%, while CG2 and SPAIN's gap are at 5%. This indeed shows that ECMP and SPAIN yield outstanding performance in a FatTree network and this is due to their capabilities in exploiting the existence of multi (redundant) paths in the network. It is important to note however that, although ECMP achieves good load balancing in the FatTree topology, it has been found to be unsuitable for Ethernet data center networks, particularly since ECMP requires IP addressing to determine the location of the destination host. This is indeed problematic in the case where the physical topology differs from the logical topology, such as the case of VLANs. Moreover, ECMP incurs latencies due to the need to determine the next-best-hop at every switch along the path to the destination host, as opposed to pre-configured paths, where the optimal trajectory from source to destination is predetermined. In addition, its memory requirement is high, since every switch needs to keep multiple paths for every pair of nodes, which can lead to oversized routing tables. Finally, ECMP was also found to be congestion-prone, since it relies

on hashing functions to determine the path to the destination host. Hashing collisions, [63] particularly between elephant flows, cause hot spots that can ultimately lead to failure in the network. Moreover, ECMP is only capable of exploring equal cost paths. Such a limitation inhibits ECMP from achieving optimal load balancing in topologies where the nodes are interconnected by multiple distinct paths of unequal cost. The interconnection of switches in the FatTree topology does not portray this limitation, which makes ECMP very well suitable for such a topology.

To confirm our observation, we study how well both ECMP and SPAIN perform in other network topologies. Namely, we consider a clique and two random topologies and the results are shown in Figures 4(c)-4(e). The figures show results obtained from ECMP, SPAIN, and CG2 as well as the LP relaxation of the CG mapping problem. We observe that, unlike in the FatTree network, the performance of both ECMP and SPAIN is far from optimal. For example, while the gap between CG2 and the lower bound optimal solution is 13% (Figure 4(c)), the gap between ECMP stands at around 40% from the lower bound. Similar findings are shown as well for random networks (Figures 4(d) and 4(e)). The rationale behind this is the fact that the clique topology, as well as the random networks, dispose more than 6 redundant paths between every pair of nodes, while ECMP was only capable of finding 1 equal cost path between every pair of nodes. This gain in the number of explored redundant paths is reflected in the better results obtained by CG2.

With respect to SPAIN, we notice inferior performance (as the figures show) in terms of achieving minimum link load for the three networks we studied. The reason is that SPAIN relies on a randomized approach when mapping flows to VLANs. This randomized approach leads to poor load balancing capabilities, especially when SPAIN exploits the existence of multiple paths with varying number of hops. Hence, a path with larger number of hops is equally likely to be utilized as a path with smaller number of hops. This leads to consuming more bandwidth from the network and yields to links with larger number of flows routed through them. Our CG2 strikes a balance between SPAIN and ECMP. It exploits the existence of multiple redundant paths by only using those paths that improve the quality of the solution and thereby achieves better load balancing. It is important to note that the authors of SPAIN mention the possibility of including a centralized control to enhance the traffic engineering capabilities of their flows to VLANs mapping algorithm. Further, to observe how the traffic is balanced throughout the network, we plot the probability density function of the link loads for R_1 (as shown in Figure 2.5). Clearly, as we have explained above, STP has the worst load distribution with CG2 showing the best link load distribution.

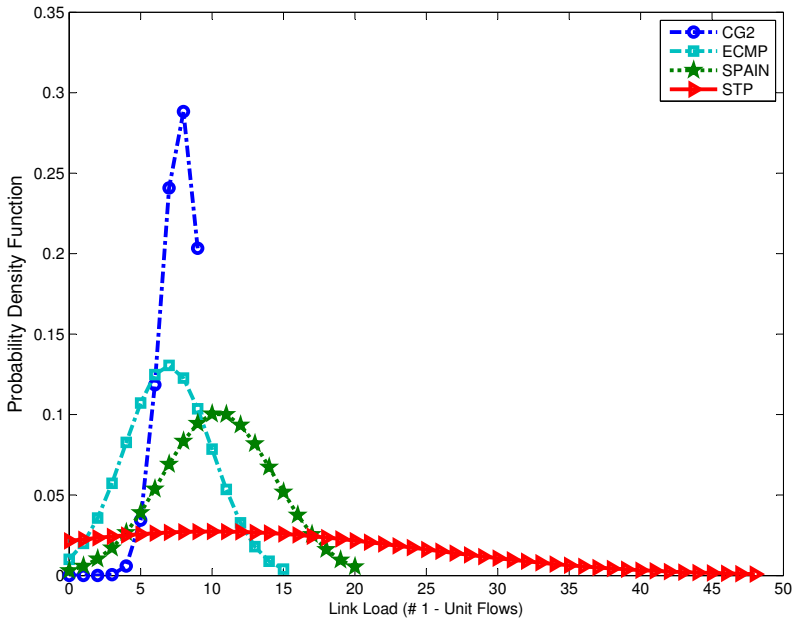


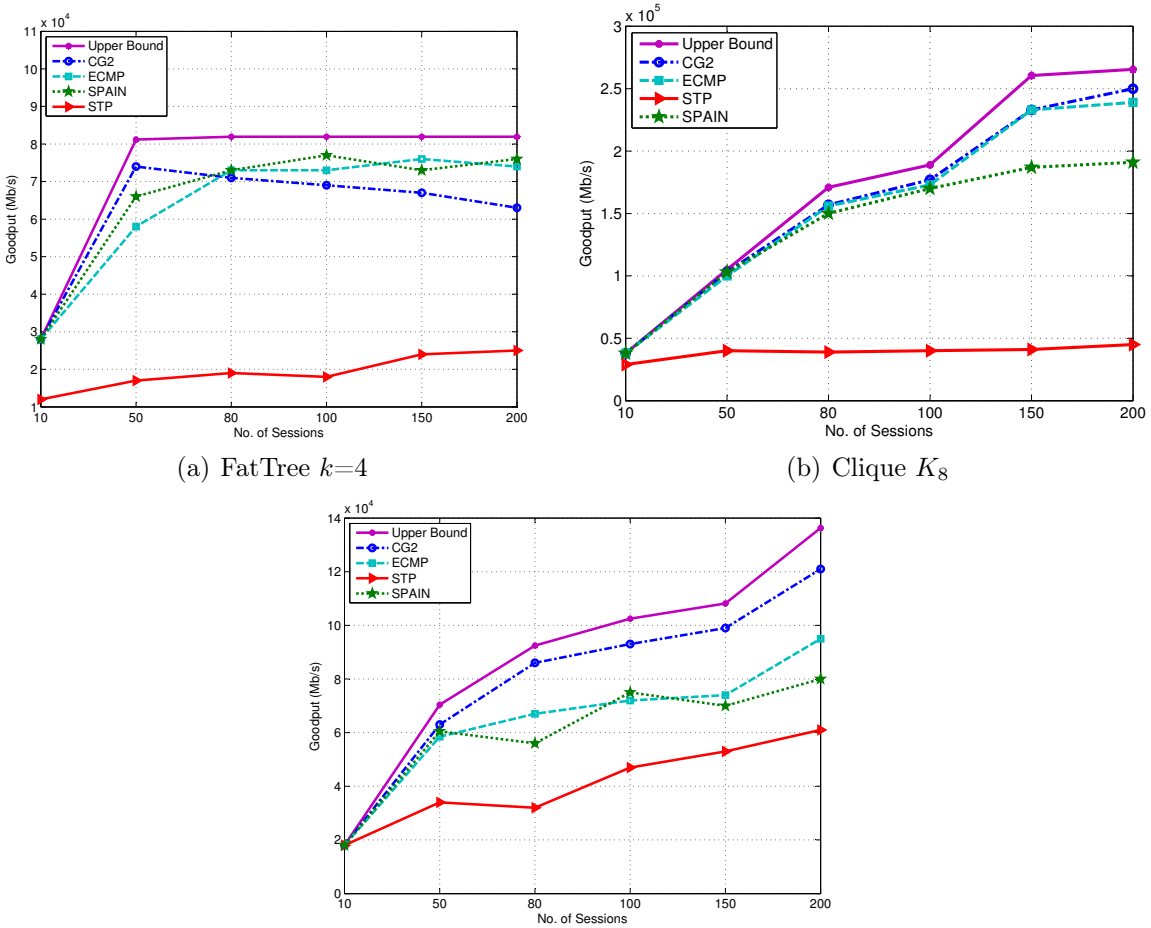
Figure 2.5: Comparison of link load between CG2, ECMP, STP and SPAIN

2.5.2.2 Goodput

In order to evaluate the network goodput, we have replaced the master's objective function by Max $\sum_v \sum_c x_c^v \, \delta_c$, and assigned a link capacity of 10Gbps and random flow demands of 1 to 3 Gbps. We aim to measure the amount of goodput provided by the CG2 model in comparison with ECMP, STP and SPAIN over a FatTree, clique and random topology. The results are shown in Figures 6(a)-6(c); the upper bound results are obtained by solving a relaxed version of the maximization problem in CG2. Clearly, for a FatTree network (Figure 6(a)), both ECMP and SPAIN achieve outstanding performance, for the same results mentioned in our previous discussions, while STP exhibits the lowest goodput. We also observe that for a clique topology (Figure 6(b)), our CG2 model outperforms ECMP and SPAIN with a 5% gap from the upper bound solution, while ECMP and SPAIN present a 7% and 13% gap respectively. Finally, in a random topology of 14 nodes and 19 links (Figure 6(c)), our CG2 model achieves a 7% overall gap, while ECMP and SPAIN provide a 22% and 25% gap, respectively.

2.6 Conclusion

In this chapter, we introduced a novel column generation approach for solving the VLAN assignment problem in cloud data centers. We presented two decomposition approaches:



(a) FatTree k=4

(b) Clique K_8

(c) Random Topology R1 - 14 switches, 19 links

Figure 2.6: Comparison of Goodput (Mb/s) between CG2, ECMP, SPAIN, and STP

an exact, as well as a semi-heuristic model to attain better runtime and scalability. We compared both models against the pure ILP model of the VLAN assignment problem, and proved that our approach yields a substantial decrease in the size of the explored search space with encouraging optimality gap. We also compared our decomposition approach against state of the art protocols in traffic engineering, our comparative analysis has shown that our model outperforms its peers in most network topologies in terms of link load, attainable gap from lower bound LP solution, as well as in goodput.

Chapter 3

Multicast Virtual Network Embedding

3.1 Problem Motivation

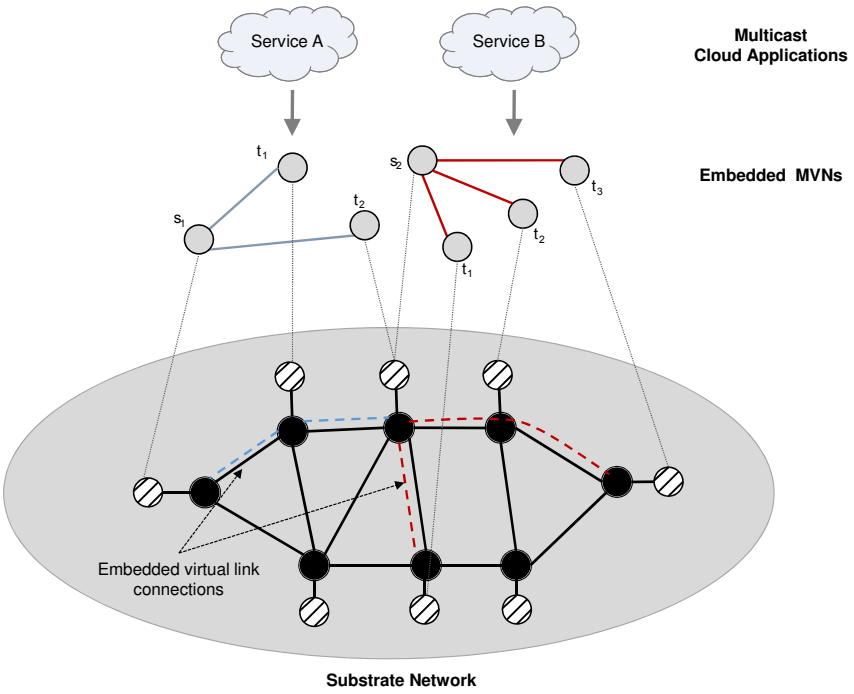


Figure 3.1: The Multicast Virtual Network Embedding Problem

Network Virtualization has been receiving significant attention for being the ideal antidote against Internet ossification [14]. Virtualization enables more freedom as heterogeneous network architectures and technologies can cohabit a shared substrate network. With server virtualization, the entire network components may be virtualized (e.g. links, routers/switches), resulting in a modular and fully isolated entity known as a virtual network. This

technology is tightly correlated with the new "philosophy" of computing as a service, in the sense that tenants applications will be hosted on large computing farms, as VNs, accessible via the Internet in a pay-per-use model.

Within this multi-tenancy environment, several challenges emerge; among these challenges is the important issue of how these VNs will coexist, also referred to as "embedded", on the same physical infrastructure. This resource allocation problem is commonly known as the NP-Hard Virtual Network Embedding (VNE) problem [33]. Hence, various efforts have been dedicated towards finding effective algorithms for solving it [14]. However, most of the existing work does not characterize the mode of communication of VN requests, assuming that they all exhibit unicast or one-to-one communication. In fact, depending on the type of service these VNs provide, communication among the participating VMs can be either unicast, multicast (one-to-many) or broadcast (all-to-all). Characterizing the type of communication in VNs is crucial for achieving optimal network operation and utilization. For instance, in unicast communication, the sender transmits data to a single receiver; thus, handling multicast communication as unicast requires transmitting multiple copies of the same data to reach each receiver. On the other hand, by handling a one-to-many communication as multicast, these multiple unicast messages can be replaced by a single multicast message, thereby incurring great benefits in terms of reducing the computation effort at the source node, reducing bandwidth consumption in the network, and subsequently ameliorating the application's throughput, and its response time [68]. Hence, for cloud providers that host multicast services with heavy traffic, it is imperative to have efficient multicast support in their data centers.

To this extent, multicast in data center networks has become a prominent research topic [69–74], with particular attention to the resource allocation problem of multicast VNs (MVNs) [69, 70]. This resource allocation problem consists of allocating physical resources to the VMs running a tenant's service, and routing the traffic flow between them via substrate paths. While this problem has been widely discussed for unicast VNs, embedding MVNs differs greatly from that of unicast for several reasons: mainly a multicast VN comprises two types of virtual nodes (machines): the multicast source node and a set of multicast recipient nodes. The traffic flow routing problem now consists of building a multicast distribution tree between the multicast source and recipients in order to avoid redundant traffic. Further, multicast services that involve real-time communication entail stringent QoS requirements, such as end-delay and delay-variation constraints. For instance, for a user-facing web-search service, if the indexing servers do not receive the search-query in a timely fashion, this

will subsequently delay the service's makespan, and hence risk violating the SLA-specified response time [75]. In addition, without a delay-variation bound, some workers (indexing servers) can lag behind waiting to receive input data, thereby slowing the pace of the entire application. This also applies to other partition-aggregate services (e.g. Map-Reduce like services for data processing) where tasks are partitioned between several machines to achieve a horizontal scalability [75]. Moreover, consider the case of a distributed database system that is consistently updated with new information. A large delay-variation between the recipient nodes that host the databases will lead to unfairness, inconsistencies, and possibly lead to incorrect computations [76], particularly for multicast-services that deal with distribution of time-critical information, such as financial service providers [68].
In light of the above, this chapter is devoted towards investigating the multicast VNE (MVNE) problem in cloud data center networks; as opposed to existing work [69, 70], here we assume that the location of the source and destination nodes in the MVN is unknown, and the communication between the source and all recipient nodes is subject to delay and delay-variation bounds. We target the case where a cloud provider wishes to host a tenant's multicast service in a data center network[1] as illustrated in Figure 3.1. Figure 3.1 illustrates an example of two multicast VNs, one running Service A, and the other running Service B, both mapped onto the same substrate network. Our main contributions can be summarized as follows:

1. We present a formal definition of the MVNE problem and state its Integer Linear Programming (ILP) formulation.

2. We provide a formal proof of the NP-Hard nature of the MVNE problem.

3. We propose a novel 3-Step approach for solving the MVNE problem, and introduce the receivers embedding problem over multicast trees.

4. We mathematically formulate the receivers embedding problem, and propose a Dynamic Programming (DP) approach for MVNs with homogeneous resource demands, that is solvable in polynomial-time over multicast trees with constant nodal degree.

5. For tree-like data center network topologies, we prove that our 3-Step MVNE with the DP for receivers embedding provides optimal solution in polynomial-time for MVNs with homogeneous resource demands.

[1]Although we consider the problem of multicast virtual networks in data centers, our work is equally applicable to a wide-area network with arbitrary substrate topologies.

6. Finally, we propose a Tabu-based search for solving the MVNE problem for multicast services with heterogeneous resource demands over arbitrary network topologies. We compare our Tabu approach against the 3-Step MVNE and other embedding heuristics, using multiple metrics and over various substrate networks. Our numerical results prove that our Tabu-based search yields high network admissibility in considerably fast runtime.

3.2 Background & Related Work

3.2.1 Multicast in Data Center Networks

Multicast in data center networks has recently become a prominent research topic [69–74, 77]. The key enabling factors for efficient multicast are related to the topological properties of data center networks and its controlled environment [78, 79], as well as the recent technological advancements pertaining to SDNs [71].

Most data center networks today adopt a multi-rooted tree topology [72, 78], which consists of several layers of commodity switches used to interconnect a large number of servers via multiple equal-cost paths. The goal of this design is to provide a high bisection bandwidth, and eliminate any network oversubscription [78]. Examples of such data centers include (among others) the Clos topology [28], and BCube [80]; some of which have already been deployed in production [15, 17]. This multi-rooted tree structure allows to overcome the scalability concerns of IP-Multicast [78, 79] in data centers. For instance, the authors in [78] developed a multicast address distribution approach that leverages the topological structure of data centers in order to circumvent the memory limitations of commodity switches, and thus expand the number of supported multicast groups. Further, the authors in [79] harness the controlled environment of data centers [17, 19] to construct reliable multicast trees that minimize packet losses in multicast communications.

Existing IP multicast routing schemes are not suited for data center networks because they do not exploit path-diversity [71], thus leading to poor resource utilization and network throughput. This is particularly true since typical data center network topologies exhibit an abundance of equal cost paths [71, 72]. Indeed, the work in [72] shows that the conventional receiver-driven multicast routing protocols yield far-from-optimal distribution trees when employed in such data center networks. Hence, they proposed an efficient and scalable source-driven multicast routing protocol that saves more than 40% network traffic compared to conventional receiver-driven routing techniques. Further, the authors in [71] leverage

SDN's global network visibility, and propose an SDN-based routing protocol for data center networks that exploits path-diversity.

SDN provides a vantage point to network and applications information, allowing the detect and handle of diverse service classes with distinct QoS requirements (e.g. delay-sensitive multicast services). By leveraging this gained knowledge about the network topology, it becomes possible to exploit path-diversity, thereby distributing the load across the network, and achieving better network throughput. In addition, the centralized control plane in SDN allows the enforcement of policies for admission control, hence it diminishes concerns of security. Further, it enables the support of multicast in commodity-switches that lack built-in support. Finally, SDN enables accurate network monitoring [81–83], thereby fostering fine-grained traffic engineering, and allowing the fulfilment of QoS requirements for delay-sensitive applications [84]. As opposed to passive network monitoring techniques that require a large hardware investment (e.g. NetFlow [85], sFlow [86]), SDN has also proved its worth [81,84,87] in performing active monitoring with high accuracy, and significantly less network overhead[2].

3.2.2 Multicast Data Center Applications

Many applications and services [72, 73, 88–90] can benefit from an efficient multicast support in data centers. For instance, High Performance Computing (HPC) applications often need to distribute a large amount of data from storage to all compute nodes [90]. HPCs are conventionally employed in distributed parallel computers such as supercomputers and grid-computing [91]. However, the emergence of cloud computing has triggered significant attention around the possibilities of migrating HPCs to the cloud [92–94]. The general consensus is that the benefits of migrating HPCs to the cloud are highly dependent on the type of workload the latter exhibits [91], [95]. Some further advocate for the benefits of a hybrid cloud/supercomputers deployment [92], [96]. Hence, in the framework of clouds, these workload-adequate HPCs can highly benefit from multicast operations to distribute the data between storage and compute nodes, while greatly alleviating redundant traffic. Similarly for web-search services, multicasting can significantly decrease the service's response time by redirecting incoming search-queries to a set of indexing servers [97]. Further, bandwidth-hungry Distributed File-Systems (DFS) are common data center applications [73], [98], [99]. DFS divides files into fixed-size chunks to be replicated and stored in different servers for

[2]The work in [84] has shown that SDN achieves 99% latency measurement accuracy with 81% less bandwidth consumption than conventional active monitoring tools.

reliability. Such type of applications can also benefit from multicasting to improve the network's throughput [72], [73]. Moreover, multicasting can greatly speed-up the distribution of executable binaries among participating servers in map-reduce [100] like cooperative computation systems [72], [88].

3.2.3 Multicast Virtual Network Embedding (MVNE)

The MVNE problem consists of allocating physical resources to multicast services with one-to-many communication mode. This problem differs from the multicast routing problem; in this latter, the location of the source and recipient nodes is known and the problem consists of finding the lowest-cost tree to interconnect them. However, in the MVNE problem the routing element interplays with the placement of virtual nodes. This reciprocal relationship between the multicast virtual node embedding and multicast routing demands separate attention, particularly when dealing with delay-sensitive multicast applications. This is true since an arbitrary node embedding solution can yield an infeasible link mapping solution, for failure to find a multicast tree that interconnects the source and the receivers with the requested QoS. One possible solution would be to try-out all feasible node mapping solutions until we find the one that renders a lowest cost distribution tree. However, for a substrate network with N nodes, and a VN request with V virtual nodes (where each virtual node has a unique resource demand), and taking the worst case where all substrate nodes have enough capacity to host any virtual node $v \in V$, there are $D = \frac{N!}{(N-V)!}$ possible node mapping solutions. Clearly enumerating all node mapping solution is an injudicious approach.

Thus, the MVNE problem has recently surfaced in the literature [69], [70]. In [70], the authors present an algorithm that maps multicast Service-Oriented Virtual Networks. In this work, the authors assume that the location of both the source and destination nodes is pre-determined, and their approach consists of finding a set of k-shortest paths between the source and each destination node that satisfy the node and link capacity constraints, while ensuring that the length of these paths respects the end-delay constraint. In [69], the authors study the problem of embedding multiple description coding-based video applications. The objective is to find various intermediate nodes between the source and multiple destinations that will perform video encoding for a given video application. It is important to note however that this work does not consider delay-sensitive applications.

Our work is different since we target the case where a tenant wishes to deploy a delay-sensitive multicast application in a cloud data center. In this case, the location of the virtual nodes (both source and receivers) in the given VN request is unknown. It is up to the cloud

provider to decide where to place the VN request, such that it optimizes his/her desired objective. The key element in this problem is identifying the placement of multicast source and receivers such that a lowest-cost distribution tree can be constructed within the requested delay-constraints.

3.3 The MVNE Problem

3.3.1 Problem Definition

In this section, we present a formal definition of the MVNE problem by describing the various components involved:

1. **The Substrate Network:** We represent the substrate network as an undirected graph, denoted by $G^s = (N,L)$, where N is the set of substrate nodes, and L is the set of substrate links. Each substrate node $n \in N$ is associated with a finite computing capacity, denoted by c_n. Similarly, each substrate link $l \in L$ has a finite bandwidth capacity, denoted by b_l.

2. **The Multicast VN (MVN) Request:** A multicast VN represents a client's request to deploy an application with one-to-many communication mode in a cloud data center. It consists of a single source node s, and a set of recipient nodes T. The source node is connected to all recipient nodes via virtual links. The set of all virtual links is denoted by E. Every virtual link $e \in E$ requires a specific amount of bandwidth, denoted by b'. For the sake of simplicity, we assume that the bandwidth demand between the source and each recipient node is the same. In addition, each virtual node is usually associated with computation demands, denoted by c'_v. We note that one of the most important properties for multicast VNs is delay; particularly for applications that involve real-time communication. Here, it is important that the source node reaches all receivers within an acceptable delay, denoted by γ. Moreover, we assume the delay variation between all recipient nodes in a given VN must also respect a given threshold δ, in order to ensure correctness and synchronization among all recipient nodes. A multicast VN is thus denoted by $G^v = (s, T, b', \gamma, \delta)$.

3. **The MVNE problem:** The MVNE problem consists of mapping the VN request onto the substrate network, such that the virtual nodes and links's resource demands are

satisfied, while making sure that the delay between the source and all recipient nodes does not violate the requested end-delay threshold. In addition, the differential delay between all the recipient nodes must also respect the delay variation constraint of the given VN request. When a multicast VN request arrives, a decision is first placed on whether to accept or reject the VN request based on the residual capacity of the substrate network, or any other admission policies dictated by the network provider. If admitted, the embedding process is initiated and decisions are made on where to place the virtual nodes and how to route the virtual links, such that the capacity constraints of the substrate network are not violated, and the desired design objectives are achieved. Note that the MVNE problem can be logically divided into two subproblems: Virtual Node Mapping (VNM) (source and recipient nodes), and Virtual Link Mapping (VLM). That latter consists of finding a MST m rooted at the source of the VN request and spans all the recipient nodes within the requested delay constraints. Hence, the MVNE problem can be formulated as follows:

Problem Definition 3.1. *Given a substrate $G^s = (N, L)$ and a multicast VN G^v = (s, T, b', γ, δ), find an optimal mapping M = (M_N,M_L) of the VN request onto the substrate, such that the demands of the virtual nodes and virtual links are satisfied, and the end-to-end delay and the delay variation constraints are met, without violating the capacity of the substrate network.*

A mapping M holds the solution for the two subproblems:

(1) Virtual Node Mapping (VNM): M_N: $(s, T) \longrightarrow N$

(2) Virtual Link Mapping (VLM): M_L: $E \longrightarrow P$; P represents the set of paths that form the multicast tree.

Theorem 3.1. *The MVNE problem is NP-Hard.*

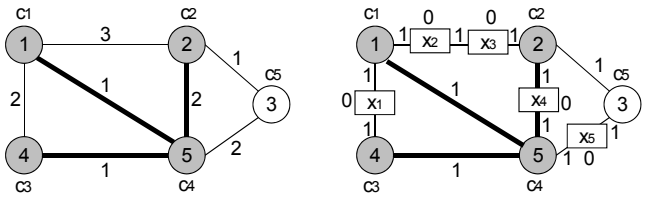
Proof. The MVNE problem can be easily seen in the NP-class, since given a solution to the MVNE problem it can be verified in polynomial-time. This can be done by first verifying the feasibility of the VNM sub-problem which takes O(|V|), where |V| = |T| + 1. Similarly, the VLM subproblem can also be verified in polynomial-time by ensuring that each path in the multicast tree respects the requested end-delay and bandwidth demands, and that the difference between the smallest and the largest path to the root satisfies the differential delay requirement.

Now, we prove that the MVNE is NP-Hard through a reduction from the unrooted K-Minimum Spanning Tree problem (K-MST) [101], which is a well-known NP-Hard problem. For the sake of completeness, we provide a formal definition of the unrooted K-MST problem [102]:

Problem Definition 3.2. *Given a substrate network $G^s = (N, L)$, where each edge $l \in L$ has a weight w_l, and a positive integer K; Find a tree of minimum weight that spans exactly K nodes.*

Our reduction consists of demonstrating a polynomial-time conversion of any instance of K-MST to an instance of MVNE, and an if-and-only-if proof that a *yes* instance of the K-MST maps to a *yes* instance of MVNE, and vice-versa. Note here, that a *yes* instance of K-MST represents a feasible solution where a minimum spanning tree with K receivers can be found in G, and a *yes* instance of MVNE represents a feasible multicast tree that spans T substrate nodes. Note here that we assume that $K = |T|$.
Consider a substrate network G^s and a virtual network G^v; without loss of generality, we assume that all the virtual nodes in G^v have a uniform CPU demand, and the bandwidth demand b' is equal to 1. Further, we let the delay and delay-variation tolerance $\gamma = \infty$ and $\delta = \infty$. Further, we assume for all nodes $n \in N$ that $c_n \geq c'_v$, $\forall\ v \in \{s, T\}$.
Now, we will convert an instance of K-MST to an instance of MVNE as follows: Given a substrate network G^s, we convert G^s to an auxiliary graph $\widetilde{G}^s = (\widetilde{N}, \widetilde{L})$, that is obtained by replacing each edge $l \in L$ connecting nodes n_1 and $n_2 \in N$, by w_l - 1 intermediate vertices of capacity 0 interconnected via edges of cost 1. For instance, if nodes n_1 and $n_2 \in N$ are connected via an edge of cost 2, this former will be replaced by a path that connects n_1 to x, and x to n_2, where x is an intermediate vertex of capacity 0, and the edges connecting x to n_1 and n_2 respectively have a cost of 1. Thus, the obtained graph $\widetilde{G}^s$ is simply G^s with two sets of substrate nodes, those that can host the virtual nodes in the given MVN request, and those that cannot (intermediate vertices of capacity 0), which reduces to our MVNE problem. Clearly this transformation can be done in polynomial time. Figure 3.2 shows the conversion of a $G^s = (N, L)$ (illustrated in Figure 2(a)) to a $\widetilde{G}^s = (\widetilde{N}, \widetilde{L})$ (illustrated in Figure 2(b)), where the link connecting substrate nodes 1 and 2 of weight 3 in G^s is replaced in $\widetilde{G}^s$ by three links of weight 1 interconnected via two intermediate nodes x_2 and x_3 of capacity 0.

Now we need to show that a *yes* instance of K-MST maps to a *yes* instance of MVNE, and vice-versa. If the $K - MST$ problem has a *yes* instance in G, this means that there exists a tree that spans K nodes with capacity $\geq c'_v$, thus the solution found for the



(a) $G^s = (N, E)$ (b) $\widetilde{G}^s = (\widetilde{N}, \widetilde{E})$

Figure 3.2: Instance of K-MST converted to an instance of the MVNE

K-MST in G forms a solution for the MVNE of G^v in $\widetilde{G}^s$. Conversely, if G^v has a feasible embedding solution in $\widetilde{G}^s$, this implies that the solution contains no intermediate nodes, since their capacity is smaller than c'_v and cannot accommodate the virtual nodes in G^v. Hence, the multicast tree that spans T receivers in $\widetilde{G}^s$ forms a K-MST in G^s, where K = $|T|$. Figure 3.2 shows how a K-MST of size 3 in G^s maps to a MVNE solution for a MVN with 3 recipient nodes in $\widetilde{G}^s$. □

3.3.2 The MVNE ILP Model (MVNE-ILP)

In this section, we mathematically formulate the MVNE problem, our objective function is to find the optimal virtual nodes mapping with the lowest-cost multicast tree that satisfies the end-delay and differential-delay constraints.

- Parameters:

$G^s(N, L)$: represents the substrate network with N nodes and L links.
$G^v = (s, T, b', \gamma, \delta)$: represents a multicast VN.
$\hat{d}_{i,j}$: represents the measured delay on link (i,j) $\in L$

- Decision Variables:

$$x_{v,n} = \begin{cases} 1, & \text{if virtual node } v \text{ is mapped on substrate node } n, \\ 0, & \text{otherwise.} \end{cases}$$

$$q^v_{i,j} = \begin{cases} 1, & \text{if link } (i,j) \text{ is chosen to reach virtual node } v, \\ 0, & \text{otherwise.} \end{cases}$$

$$z_{i,j} = \begin{cases} 1, & \text{if link } (i,j) \text{ is part of the multicast tree,} \\ 0, & \text{otherwise.} \end{cases}$$

$t_{i,j}$: represents the traffic flow on link (i, j).
θ_{min}, θ_{max} : represents the minimum and maximum delay.

- Mathematical Model:

$$Min \sum_{(i,j)\in L} t_{i,j}$$

Subject to

$$\sum_{n\in N} x_{v,n} = 1 \quad \forall v \in \{s,T\} \tag{3.1}$$

$$\sum_{v\in\{s,T\}} x_{v,n} \leq 1 \quad \forall n \in N \tag{3.2}$$

$$\sum_{v\in\{s,T\}} x_{v,n}.c'_v \leq c_n \quad \forall n \in N \tag{3.3}$$

$$\sum_{j:(i,j)\in L} q^v_{i,j} - \sum_{j:(j,i)\in L} q^v_{j,i} = x_{v,i} - x_{s,i} \quad \forall i \in N, v \in T \tag{3.4}$$

$$\sum_{(i,j)\in L} q^v_{i,j}\hat{d}_{i,j} \leq \gamma \quad \forall v \in T \tag{3.5}$$

$$\theta_{min} \leq \sum_{(i,j)\in L} q^v_{i,j}\hat{d}_{i,j} \quad \forall v \in T \tag{3.6}$$

$$\theta_{max} \geq \sum_{(i,j)\in L} q^v_{i,j}\hat{d}_{i,j} \quad \forall v \in T \tag{3.7}$$

$$\theta_{max} - \theta_{min} \leq \delta \tag{3.8}$$

$$z_{i,j} \geq q^v_{i,j} \quad \forall v \in T,\ (i,j) \in L. \tag{3.9}$$

$$t_{i,j} = z_{i,j}.b' \quad \forall (i,j) \in L. \tag{3.10}$$

$$t_{i,j} \leq b_{i,j} \quad \forall (i,j) \in L. \tag{3.11}$$

$$\sum_{i\in S}\sum_{j\in S} z_{i,j} \leq |S| - 1 \quad \forall S \subset N, 2 \leq |S| \leq N \tag{3.12}$$

Constraint (3.1) indicates that each virtual node in the given MVN must be mapped to a single substrate node, while Constraint (3.2) ensures that each virtual node is mapped to a distinct substrate node. Constraint (3.3) represents the substrate nodes capacity constraint. Constraint (3.4) represents the flow conservation constraint, and Constraints (3.5)-(3.8) ensure that the constructed multicast tree satisfies the end-delay and delay-variation constraints. Constraint (3.9) indicates the distinct substrate links that form the constructed multicast tree. Constraint (3.10) measures the traffic provisioned for each link in the constructed multicast tree, and Constraint (3.11) ensures that the provisioned traffic does not violate the substrate links capacity. Finally, Constraint (3.12) represents the subtours elimination constraint.

Note that here, $\hat{d}_{i,j}$ represents the measured latency at every link $(i,j) \in L$, which is the sum of the queueing delay, propagation delay, transmission delay, and processing delay. Hence, the end-delay between the source s and any receiver t is equal to $\sum_{l \in P_{(s,t)}} \hat{d}_{i,j}$, where $P_{(s,t)}$ represents the physical path between the host of s and t respectively. Throughout this work, we assume that the delay experienced on each edge in the substrate network is an input, which can be obtained via network monitoring tools[3].

Clearly, the proposed MVNE-ILP model is hard to scale given that the MVNE problem is NP-Hard. To this extent, we propose a novel 3-Step approach for solving it. Our 3-Step MVNE approach tackles the MVNE problem disjointly, by first finding a set of feasible multicast trees, and then attempts to map the recipient nodes onto these trees.

3.4 The MVNE Heuristic (MVNE-H)

3.4.1 The 3-Steps MVNE Heuristic with a Node Mapping Model (MVNE-HNM)

Our approach comprises of 3 steps, as illustrated in Algorithm 3.1. It begins by first pruning the substrate network in order to identify the set of eligible nodes that can become root nodes in a multicast tree. This pruning results in a list of eligible root nodes. Next, from each one of these root nodes, multicast trees are constructed and placed in a dedicated set which will be fed to an ILP node mapping model in order to return the optimal mapping solution within this given set.

[3]There are numerous network monitoring tools that can be used to perform active or passive data centers traffic monitoring [82–84]. Further, SDN-enabled latency measurement tools have been proven to achieve 99% accurate measurement with significantly low network overhead [84, 87]

Algorithm 3.1 The MVNE Heuristic Algorithm

Given
$G^s = (N, L)$ /*an arbitrary topology*/
$G^v = (s, T, b'_v, \gamma, \delta)$

Step 1: Prune the Network
$R = \{\ \}$; /*Initialize the set of Eligible Root Nodes*/
for $n \in N$ **do**
    if $(c_n \geq c'_s)$ **then**
        $R = R \cup n$;
    end if
end for

Step 2: Build Multicast Trees
$\mathcal{M} = \{\ \}$; /*Initialize the set of Multicast Trees*/
for $r \in R$ **do**
    $m = \text{DFS}(r,\gamma)$;
    if $(\text{CountEligibleNodes}(m) \geq T)$ **then**
        $\mathcal{M} = \mathcal{M} \cup m$;
    end if
    $m' = \text{BFS}(r,\gamma)$;
    if $(\text{CountEligibleNodes}(m) \geq T)$ **then**
        $\mathcal{M} = \mathcal{M} \cup m'$;
    end if
end for

Step 3: Perform Node Mapping
Return NodeMappingModel(G^s, G^v, $\mathcal{M}$);

1. **Step 1 - Prune the Network**

   Given a VN request, the process begins by pruning the network to find a set of substrate nodes that can become the root of a multicast tree. This is done by selecting the substrate nodes which satisfy the resource demands of the given VN. At the end of Step 1, the process returns a list of eligible root nodes. This pruning process takes $O(n)$ time.

2. **Step 2 - Find Subgraphs with Potential Feasible Solution**

   The second step consists of building multicast trees, each rooted at one of the nodes in the list of eligible roots provided by Step 1. Building a multicast tree involves multiple parameters, particularly when it is used to support real-time communications that are delay-sensitive. The first concern is cost, which is represented in the number of links used in the resultant distribution tree (since it is proportional to the amount of traffic to-be provisioned for this MVN). Given the list of potential root nodes, the process selects a node from the list, and then expands it into a tree using the Depth First Search (DFS) algorithm. The DFS allows us to maintain the depth of the tree within the range of γ, that is ensure that the sum of the delays experienced along any path in the constructed tree satisfies the end-delay constraint. Once the depth of any branch (path) in the tree exceeds γ, this branch seize to grow. In order to avoid multicast trees that will definitely lead to infeasible solutions, a basic check is performed on each tree to count the number of eligible substrate nodes that can host the recipient nodes of the given VN. If the number of eligible nodes is larger than the number of recipient nodes in the VN request, the multicast tree is added to a dedicated set $\mathcal{M}$. The same process is performed for every node in the list of potential root nodes. Note that our process expands the tree in a source-driven manner by building on the findings in [72], where source-driven multicast trees were found to be more suitable for the topological architecture of the new generation data center networks. In order to further diversify the set of multicast trees, the process runs, again, using Breadth First Search (BFS) in order to construct the shortest path multicast tree from every node in the list of potential root nodes. Note that the depth of the BFS trees is also limited to γ, and a basic check on the number of eligible nodes is also performed. At the end of Step 2, the process returns a set of potential subgraphs (multicast trees), which will be used in Step 3 to perform the recipient nodes mapping. Step 2 has a complexity of $O(2N.(N + L))$, since at worst, all the nodes in the substrate network might be in the list of eligible root nodes, and at each eligible root node, 2 multicast trees will be

constructed with a worst case complexity of $O(N + L)$.

3. **Step 3 - Perform Node Mapping** Given a set of multicast trees, where each tree exhibits a set of eligible substrate nodes, the question becomes: which tree to select, and how to map the recipient nodes of the given VN request. We formulate an ILP model with the objective to balance the load in the network. The model is performed online upon the arrival of each VN request. It assumes as input the set of multicast trees, the eligible substrate nodes in each multicast tree and the end-delay experienced at every path from the root node to any eligible substrate node. The model returns the optimal multicast tree with a recipient node mapping solution that respects the delay-variation constraint of the given VN. The mapping of recipient nodes can be formulated as follows :

The Node Mapping Model :

- Parameters:

$\lambda_{n,m} = \begin{cases} 1, & \text{if node } n \text{ belongs to multicast tree } m, \\ 0, & \text{otherwise.} \end{cases}$

$\varphi_{(n,m)}$: delay of the path $P_{r,n}$, where r is the root of multicast tree m.

$\omega_{(i,j)}^{n,m} = \begin{cases} 1, & \text{if } (i,j) \text{ is in the path from the root of } m \text{ to } n, \\ 0, & \text{otherwise.} \end{cases}$

ρ: is an input parameter between 0 and 1 that allows to adjust the importance of load balancing between the substrate nodes and links.

- Decision Variables:

α: represents the maximum node residual capacity.

β: represents the maximum link residual capacity.

$x_{v,n}^{m} = \begin{cases} 1, & \text{if } v \text{ is mapped onto node } n \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$

$y_m = \begin{cases} 1, & \text{if multicast tree } m \text{ is chosen,} \\ 0, & \text{otherwise.} \end{cases}$

τ_n: represents the residual capacity of physical node n.

$\tau'_{i,j}$: represents the residual capacity of physical link (i,j).

- Mathematical Model:

$$Max\ [\rho\alpha + (1-\rho)\beta]$$

Subject to

$$\sum_{m\in\mathcal{M}}\sum_{v\in\{s,T\}} x_{v,n}^{m} \leq 1 \quad \forall n \in N \tag{3.13}$$

$$\sum_{m\in\mathcal{M}}\sum_{n\in N} x_{v,n}^{m} = 1 \quad \forall v \in T \tag{3.14}$$

$$x_{v,n}^{m} \leq \lambda_{n,m} \quad \forall v \in T, n \in N, m \in \mathcal{M}. \tag{3.15}$$

$$\sum_{m\in\mathcal{M}} y_m = 1 \tag{3.16}$$

$$x_{v,n}^{m} \leq y_m \quad \forall v \in T, n \in N, m \in \mathcal{M}. \tag{3.17}$$

$$x_{s,r}^{m} = y_m \quad \forall m \in \mathcal{M}. \tag{3.18}$$

$$\sum_{m\in\mathcal{M}}\sum_{v\in T} x_{v,n}^{m} c_v' \leq c_n \quad \forall n \in N. \tag{3.19}$$

$$\tau_n = c_n - \sum_{m\in\mathcal{M}}\sum_{v\in\{s,T\}} x_{v,n}^{m} c_v \quad \forall n \in N. \tag{3.20}$$

$$\tau_n \geq \alpha \quad \forall n \in N. \tag{3.21}$$

$$\theta_{min} \leq \sum_{m\in\mathcal{M}}\sum_{n\in N} x_{v,n}^{m} \varphi_{(n,m)} \quad \forall v \in \{T\}. \tag{3.22}$$

$$\theta_{max} \geq \sum_{m\in\mathcal{M}}\sum_{n\in N} x_{v,n}^{m} \varphi_{(n,m)} \quad \forall v \in \{T\}. \tag{3.23}$$

$$t_{i,j} = \sum_{m\in\mathcal{M}}\sum_{n\in N}\sum_{v\in T} x_{v,n}^{m} \omega_{(i,j)}^{n,m} \quad \forall (i,j) \in L. \tag{3.24}$$

$$\tau'_{i,j} = b_l - z_{i,j}b' \quad \forall l : (i,j) \in L. \tag{3.25}$$

$$\tau'_{i,j} \geq \beta \quad \forall (i,j) \in L. \tag{3.26}$$

The objective function is to balance the load in the substrate network in order to achieve a better acceptance ratio on the long-run, hence by maximizing the residual capacity of substrate nodes and links, we can achieve our load balancing objective. Constraint (3.13) indicates that for a given MVN, at most one virtual node can be mapped onto one physical node. Constraint (3.14) ensures that every virtual node, within a given MVN, must be mapped onto a single physical node; in the case where at least one virtual node could not be mapped (e.g. lack of resources), the model will return an infeasible solution. Constraint (3.15) indicates that a virtual node v can be mapped on a substrate node n in a multicast tree m, if and only if, node n belongs to the multicast tree m. Constraint (3.16) forces a single multicast tree to be selected among the set of multicast trees $\mathcal{M}$. Constraint (3.17) ensures that a virtual node v cannot be mapped on a multicast tree m, if multicast tree m is not chosen. Constraint (3.18) indicates that the source node s of the given VN must be mapped onto the root node r of the chosen multicast tree. Constraint (3.19) represents the substrate node capacity constraints. Constraint (3.20) measures the residual capacity of the substrate nodes, and constraint (3.21) enforces the minimum residual node capacity to be larger than α. Constraint (3.22) and (3.23), along with Constraint (3.8) enforce the differential delay variation constraint. Constraint (3.24) measures the traffic flow on link (i,j). Constraint (3.11), (3.25) and (3.26) are used to measure the residual capacity of substrate links, and set the residual capacity of all substrate links to be larger than β.

3.4.2 A Dynamic-Programming approach for solving the recipient node mapping problem over a multicast tree

Problem Definition 3.3. *Given a VN request G^v = (s, T, b', γ, δ) and a multicast tree m, where s is mapped at the root r of m; find an optimal mapping of the recipient nodes T onto the substrate nodes in m, such that we obtain the lowest cost tree that satisfies the resource demands of all recipient nodes, and respects the delay-variation constraint.*

Here, we assume that all recipient nodes have the same resource requirements (e.g. CPU demands), and that all the substrate nodes in m have enough capacity to host any recipient

node $t \in T$. Below we provide the procedural details of our DP approach, and then provide a formal proof of its polynomial runtime for multicast trees with constant nodal degree. Our DP approach works as follows: First, we start by dividing the tree into subtrees, where the difference between the shortest and the longest end-delay experienced between any pair of nodes satisfies the delay-variation constraint. Hence, given a tree $m \in \mathcal{M}$ of height γ (since any branch in the multicast tree is bounded by the end-delay constraint), there will be at most (γ - δ) partitions, each respecting the delay-variation constraint δ. We denote this set of all partitions as $\hat{P}_m$.

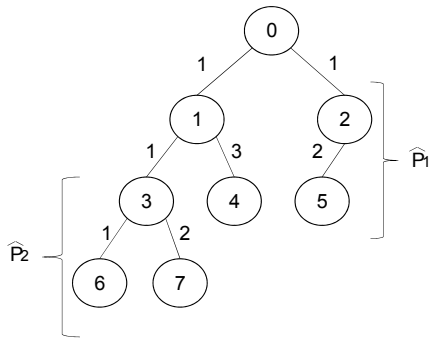
Theorem 3.2. *A feasible node mapping solution of T recipient nodes in a partition $\hat{p} \in \hat{P}_m$ is also a feasible node mapping solution for the multicast tree m.*

Proof. The goal of partitioning the multicast tree m is to find all node mapping solutions where all T receivers can be mapped within a single partition. Given that the differential-delay experienced between any pair of nodes in a subtree respects the delay-variation constraint, then the resultant node mapping solution in each partition will never violate that constraint. Further, since there exist a single path from the root(s) of each partition $\hat{p} \in \hat{P}_m$ to the root of the multicast tree m, this implies that the differential-delay of any pair of paths connecting a pair of host nodes to the root(s) of a partition $\hat{p}$ remains the same when measured towards the root of the multicast tree m. □

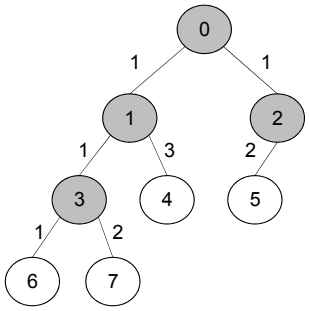
Let $U_{\hat{p}}$ denote the set of substrate nodes in partition $\hat{p}$. For each partition $\hat{p}$, we define $C(k,u)$ to be the cost of mapping k recipient nodes ($1 \leq k \leq |T|$) on the partition rooted at substrate node u, $\forall\ u$ in $U_{\hat{p}}$. This consists of mapping a recipient node on substrate node u, and finding the minimum cost of distributing the $k-1$ remaining nodes on u's children. We denote the set of u's children as H_u, and the number of nodes in the subtree rooted at u as H. Also, let $\hat{S}$ be the set of all partitions of $k-1$ receivers among all, or a subset, of H_u. We define the following recurrence for $C(k,u)$:

$$C(k,u) = \begin{cases} 0, & k = 1, \\ \infty, & k > H, \\ min\{\sum_{h_i \in \hat{s}} (w(u,h_i) + C(n_i,h_i))\} \forall \hat{s} \in \hat{S}, \sum_{hi \in \hat{s}} n_i = k-1, & 1 < k \leq |T| \end{cases} . \tag{3.27}$$

Here, $w(u,h_i)$ represents the weight of the edge connecting u to its child h_i in the subtree $\hat{p}$. The aim of the DP algorithm is to find the mapping that leads to the tree with the lowest total edge weights.



(a) Multicast Tree with Partitions



(b) Optimal Solution

K \ U	1	2	3	4	5
K=3	4	∞	∞	∞	∞
K=2	1	2	∞	∞	∞
K=1	0	0	0	0	0

(c) Matrix A for $\hat{P}_1$

K \ U	3	4	5	6	7
K=3	3	∞	∞	∞	∞
K=2	1	∞	∞	∞	∞
K=1	0	0	0	0	0

(d) Matrix A for $\hat{P}_2$

Figure 3.3: Illustration of the DP Approach

Now, for each subtree $\hat{p} \in \hat{P}$, we start by computing $C(k, u)$ at the leaf-nodes and moving upwards until reaching the highest level in the subtree. The computed cost values in each subtree will be stored in a dedicated matrix $A[T][U]$, which enables memoization as computing the cost at a particular node will make use of the cost values computed for its children, hence the benefit of adopting a bottom-up computation.

Finally, the optimal solution consists of finding the lowest cost (C^*) of embedding T receivers on any subtree $\hat{p} \in \hat{P}$. This can be obtained using the following equation:

$$min\{\sum_{u \in \hat{s}}(w(u,r) + C(k_u, u))\}, \forall \hat{s} \in \hat{S}_{\hat{p}}, \forall \hat{p} \in \hat{P}, \sum_{u \in \hat{s}} k_u = |T| \tag{3.28}$$

$w(u, r)$ is the weight of the edge connecting any node $u \in \hat{p}$ to the root r of the multicast tree m. Note here that if the set of nodes $u \in \hat{p}$ share common links to the root r, these links will be counted only once. Hence, the optimal solution is about finding the subtree $\hat{p}$ of m that contains the partition $\hat{s}$ with the lowest cost of embedding all T receivers. Figure 3.3 illustrates through an example our DP approach. Consider the multicast tree m shown in Figure 3(a), where the number next to each link represents the weight. Now, consider that we are looking to embed a 3-receivers MVN atop this tree, with an end-delay of 2 and a differential-delay of 1. For the sake of simplicity, in this illustrative example we assume that all links exhibit a uniform delay measurement, hence γ = height of the tree

= 3. Subsequently, we find that there are γ-δ = 2 partitions (as shown in Figure 3(a)), that we denote as $\hat{P}_1$ and $\hat{P}_2$, respectively. Next, we apply our DP recurrence to compute matrix $A[T][U]$ for $\hat{P}_1$ and $\hat{P}_2$, as shown in Figures 3(c) and Figure 3(d), respectively. For instance, the cost of placing two receivers on node 1 in partition $\hat{P}_1$ is equivalent to C(2,1) = min{(w(1,3)+C(1,3)), (w(1,4)+C(1,4))} = min {(1),(3)} = 1. Finally, we return the partition that yields the lowest cost distribution of T receivers in m (as shown in Figure 3(b)), which in this case is partition $\hat{P}_1$ with 2 receivers embedded on the sub-tree rooted at 1 ($C(2,1)$), and 1 receiver placed on the subtree rooted at 2 ($C(1,2)$), with a total cost C^* = $C(2,1) + w(0,1) + C(1,2) + w(0,2) = 3$.

Theorem 3.3. *For any multicast tree m with constant nodal degree $\tilde{d}$, the DP approach is polynomial in the size of the multicast tree $|U|$.*

Proof. Our DP consists of examining at each substrate node $u \in U_{\hat{p}}$ the lowest possible cost of placing a single virtual node at u and the remaining $T-1$ virtual nodes on u's children. Finding the lowest cost placement thus consists of finding the optimal partition of $T-1$ virtual nodes on u's children. Enumerating the set of all partitions can be seen as the problem of enumerating the different placements of K identical objects into n distinguishable boxes. This former can be obtained using the following binomial coefficient $\binom{n+k-1}{n}$. Hence, by considering the $|T-1|$ receivers to be the objects and the set of u's children $|H_u|$ to be the boxes, at each node u the number of partition is equal to $\binom{|H_u|+|T-1|-1}{|H_u|}$; given that such examination must occur at each node u in the subtree $\hat{p}$, then the complexity of running the DP algorithm in each subtree $\hat{p}$ is $O(|U_{\hat{p}}|\binom{\tilde{d}+|T|-2}{\tilde{d}})$, where $|U_{\hat{p}}| \leq N$. Thus, we can deduce that the worst-case runtime of our DP algorithm is $O((\gamma\text{-}\delta).|N|.\binom{\tilde{d}+|T|-2}{\tilde{d}})$, which is equivalent to $O((\gamma-\delta).|N|.\widetilde{d^T})$. Hence, if the nodal degree in the multicast tree is constant, then the number of partitions at each node is also constant, which renders a polynomial-time DP approach in the size of the network. □

3.4.3 Optimality Analysis

Recall that our 3-Step MVNE receives a subset of trees enumerated using BFS and DFS methods, on top of which the receivers embedding model is employed to return the lowest cost tree that can host all receivers within the differential delay constraint. This implies that if our 3-Step MVNE was fed with the set of all multicast trees, it will return the optimal solution. However, finding all multicast trees in an arbitrary graph can be a tedious task, particularly when there are an exponential number of them. For instance, embedding a

MVN with loose delay constraints in a complete graph consists of enumerating the set of all spanning trees (since all substrate nodes can be reaches within this loose end-delay threshold). In a clique topology, there are n^{n-2} spanning trees, which is definitely exponential. However, it has been shown that multi-rooted tree like data center topologies [19], such as the FatTree network, can be leveraged to construct Minimum Steiner Trees (MSTs) in polynomial time [71]. This is achieved by fixing the designated switch at every layer, thereby minimizing the number of intermediate (Steiner nodes) switches used to interconnect a set of receiver points. For instance, using this approach, we can find at most $\frac{K^5}{16}$ MSTs in a FatTree topology, where $\frac{K^3}{4}$ represent the number of all candidate source nodes (equivalent to the number of server racks), and $\frac{K^2}{4}$ is the number of distinct MSTs that can be found from a particular source node to all other server racks. Clearly, finding a feasible receiver embedding solution for a given MVN atop one of these MSTs will definitely be the optimal MVNE solution for this MVN. This is true since any other embedding solution found outside this set of MSTs will definitely have a higher cost. Now, if no feasible receiver embedding solution can be found in this set, then it is possible that a feasible solution may exist outside this set. This is because MSTs do not necessarily guarantee the satisfaction of the differential-delay among the various receivers.

Thus, one needs to enumerate a larger subset of trees to guarantee finding a solution for delay-sensitive MVNs. Note that, here we are not interested in trees that span all nodes in the network, but only those that span the server racks where the receivers embedding will take place. Enumerating these spanning trees represents the worst case where all server racks have enough residual capacity; however, note that any branch(es) that violate the end-delay constraint will be automatically pruned out. To this extent, we embarked on enumerating the number of all spanning trees in a FatTree network. The combinatorial arguments of our tree enumeration are provided in Appendix 9.1.

Subsequently, it is found that a FatTree (k=4) holds 128 spanning trees, whereas a FatTree(k=8) contains roughly 1710^9 trees, which clearly indicates that the number of spanning trees grows exponentially fast in the size of the network switches.

Let P^* represent the set of all MSTs, and P the set of all spanning trees, $P^* \subseteq P$. In addition to the fact that P is exponentially large, many of the trees contained in P are too costly to select as they exhibit a large number of Aggregation Switches (ASs) and Core Switches (CSs) (see Figure 5.4). In fact, in a FatTree network, the more ASs and CSs are chosen, the more up-links the distribution tree will include (see Figure 5.4). This number of up-links reflects the amount of redundant traffic that will be endured by the underlying network, which will

subsequently decrease the network's throughput, and dash any advantage of multicasting. Hence, it is important to place a limit on the number of ASs and CSs a source node can use to reach other receivers, and reject any solution that surpasses this limit. Let c_1 denote the maximum number of ASs, and c_2 be the maximum number of CSs (c_1 and c_2 are predetermined by a policy selected by the network operator). Hence, the number of spanning trees that can be formed with at most c_1 and c_2 ASs and CSs, respectively, will drastically decrease, and will thus become independent of the size of the network. We denote the set of spanning trees under the above policy as $\widetilde{P}$ ($P^* \subseteq \widetilde{P} \subseteq P$).

Theorem 3.4. *Any optimal solution returned by our 3-Step MVNE taking as input $\widetilde{P}$, is also global optimal over P.*

Proof. $\forall\ \widetilde{p} \in \widetilde{P}$, $\widetilde{p}$ has a lower cost than any tree $\in (P\text{-}\widetilde{P})$. That is because any other tree $p \in (P\text{-}\widetilde{P})$ has more Steiner points, which directly implies a larger number of up-links. Thus, if our 3-Step MVNE finds the optimal solution in the subset $\widetilde{P}$, then this solution is definitely global optimal since it represents the lowest cost solution that hosts all receivers while satisfying the end-delay and differential delay constraint. Indeed, any other solution that will be found outside this subset may also satisfy the delay constraints, but it will definitely be more costly to choose as it includes more Steiner points. □

It is important to note that when no solution can be found within $\widetilde{P}$, then there could have been a way to find a feasible solution by incurring more Steiner points. However, such a solution (if found) is deemed as not cost-effective under the chosen policy by the network operator as it incurs significant redundant traffic and decreases the network's throughput. Hence, the network provider may either opt to increase the size of c_1 and/or c_2, or refrain from admitting these requests.

Preposition 3.1. *In a FatTree network, the 3-Step MVNE with the DP for receivers embedding returns optimal solution in polynomial time.*

Given the set $\widetilde{P}$, which is independent of the size of the network, this set will be passed on to our DP, that will run the receivers embedding over each tree in $\widetilde{P}$, and return the one that satisfies the delay constraints. Given that the degree in the FatTree network is always constant, indicated by the number of ports in the Ethernet switches K, then our DP approach will run in polynomial-time for each tree. Here some slight modifications need to be performed on our current DP recurrence, since for multicast trees built over the FatTree network, the embedding can only be performed on leaf nodes. Here, we denote H_u to be

the leaf nodes in the subtree rooted at u. Hence, our recurrence can now be formulated as follows:

$$C(m,u) = \begin{cases} 0, & m = 1 \;\; \&\& \;\; |H_u| = 0 \\ \infty, & m > |H_u|, \\ min\{\sum_{h_i \in \hat{s}}(w(u,h_i) + C(n_i,h_i))\} \forall \hat{s} \in \hat{S}, \sum_{hi \in \hat{s}} n_i = k, & 1 < k \leq |T| \end{cases} .$$
(3.29)

Recall that our DP has a complexity of $O((\gamma - \delta).|N|.(\widetilde{d}^T))$, where $\widetilde{d} = K$ in the FatTree network.

Hence, we can conclude that in the FatTree Network[4], our 3-Step MVNE with DP for receivers embedding can return the optimal solution in polynomial-time of the size of the network under the two following constraints: The set of multicast trees $\widetilde{P}$ have a limit on the number of up-links, otherwise they incur significant redundant traffic. The MVNs are assumed to have uniform resource demands. Under these constraints, our proposed method guarantees optimality in polynomial-time.

3.5 Tabu-based approach for solving the MVNE problem (MVNE-Tabu)

3.5.1 Tabu Search Algorithm

Tabu search [103] is a widely adopted meta-heuristic algorithm, which was proven capable of achieving optimal and near-optimal solutions for scheduling problems and various optimization problems in the area of telecommunication. The most attractive feature of Tabu-based search is the use of "adaptive memory" to direct the search towards solutions that best service the desired objective function. Its success lies in its ability to allow solutions with degraded performance, and hence escape local optimum. An efficient Tabu implementation highly depends on 4 main ingredients: first, the neighborhood structure, which consists of a set of moves within a neighborhood that are assessed based on a tailored cost function. At each iteration of tabu, a new move is selected in order to explore the search space by jumping from one solution to another. Second marking moves as Tabu prevents search loops;

[4]Note that, the same analysis is applicable to any other multi-rooted tree-like data center network topology with constant nodal degree, e.g. Portland [19] and VL2 [17].

i.e. picking a move that shifts back to a previously explored solution. Third, finding a good initial solution paves the way towards finding a better final solution. And finally, aspiration and diversification strategies, where an aspiration criteria allows tabu moves to be reconsidered if these latter lead to the best known solution, whereas diversification strategies enable the search of unexplored areas in the search space.

3.5.2 Intuition of the Tabu-Search Algorithm

We have tested our MVNE-HNM on a FatTree (k=8) (#Hosts = 128, #Links = 256) for VNs with size varying between 3 to 12 recipient nodes. We assume that the VNs arrival follows a poisson process with an arrival rate λ per time unit, and the departure follows a negative exponential distribution with a service rate μ per time unit. Hence, $\frac{\lambda}{\mu}$ represents the load. We randomly generated a set of 100 VNs that follow a poisson distribution with a normalized load of 4, and we found that our MVNE-HNM took almost 12 hours to embed all 100 VNs. That is an average of 7 minutes to embed each incoming VN. For an online algorithm with highly interleaving VNs, 7 minutes could be quite costly. The slow runtime of the MVNE-HNM is mainly caused by the ILP nature of the node mapping mode. ILPs are known to be hard to scale. In this regard, we propose two different node embedding heuristics to replace the ILP model : A greedy node mapping approach, and a First Fit embedding, that we denote as MVNE-HG and MVNE-HFF, respectively. The greedy node mapping mainly consists of sorting the candidate substrate nodes in a given multicast tree based on their CPU capacity, and then iteratively start to map the receiver with the highest CPU requirement to the substrate with the highest CPU capacity. Similarly, First Fit follows the same approach, but instead it sorts the candidate substrate nodes based on their proximity to the root node, in an attempt to yield the lowest cost multicast tree. Throughout our numerical results, we find that though these two heuristics yield outstanding performance in terms of computational time, they fail to embed multicast VNs that are highly delay sensitive due to the fact that they are completely oblivious to the differential-delay constraint. Hence the need for a heuristic MVNE embedding technique that is both scalable and delay-aware. This motivates us to propose a MVNE technique that adopts a Tabu-based search. In what follows, we identify each one of these components towards introducing our MVNE Tabu-based algorithm.

3.5.3 Delay-Aware Tabu-based Algorithm for MVNE

Our Tabu-based approach is performed online upon the arrival of each new multicast VN request. It consists first of building an initial node mapping solution in a greedy fashion. This is done by sorting the substrate nodes in descending order of their resource capacity, and the virtual nodes in descending order of their resource demands. Next, we try to map the virtual node with the highest resource demands to the substrate node with the highest capacity; while doing so, a preliminary test on the bandwidth capacity of the substrate node's adjacent links is performed. This helps eliminate node embedding that will yield an infeasible link embedding. This greedy-embedding is performed iteratively until each virtual node in the given VN request is mapped to a substrate node. Should the initial node embedding fail to yield a feasible solution, then we can conclude that the given substrate network cannot admit this multicast VN, and the process terminates.

Upon obtaining an initial node mapping solution, an initial link mapping solution is built by running a shortest-path algorithm (e.g. Dijkstra) between the host of the multicast source node to all hosts of the multicast recipient nodes. Subsequently, we obtain an initial mapping solution (composed of an initial node mapping and link mapping solutions). Next, we define a move to be a shift of a virtual node (either source or receiver) from the current host to a new substrate node. This move is accompanied by redrawing the shortest-path from the new host to the rest of the multicast tree. The neighborhood structure thus consists of finding, for each virtual node v, the set of p candidate hosts (moves) $\hat{C}_v$ within k hops from its current host, and which satisfies its CPU demands. The union of all candidate sets $\hat{C}_v$, $\forall\ v \in V$, is denoted as $\hat{C}$. Subsequently, the cost of each candidate move is evaluated using the following equation: $\tilde{M}_{\hat{c}} = L + \tilde{P}_1 + \tilde{P}_2$; where L is the number of distinct links in the multicast tree, $\tilde{P}_1$ is the penalty for violating the end-delay constraint, and $\tilde{P}_2$ is the penalty for violating the differential-delay constraint. Here, $\tilde{P}_1$ and $\tilde{P}_2$ consists of multiplying each unit of end-delay violation by a constant c_1, and each unit of differential-delay violation by another constant c_2, respectively. The penalties c_1 and c_2 must be set to some values high enough in order to prevent Tabu from selecting moves with delay violations (in our experimental results, we set c_1 to 100 and c_2 to 50). The move with the lowest cost is deemed as the best candidate move, and the current solution is updated by shifting the mapping of virtual node v, of the best candidate move, to its new host node. In addition, the paths from v's new host to the multicast tree is updated by running the short-path algorithm from the new host node to all other substrate nodes hosting virtual nodes with whom v communicates. If v is a recipient node, this means that only the path to the source node is updated, whereas in the case where

v is the root, then the paths to all recipient nodes will be updated.

We maintain two Tabu lists, one for virtual nodes, and another for substrate nodes. Each time a new move is made, the shifted virtual node v is marked as Tabu, as well as the substrate node n that was hosting v. This prevents cycling back to an old solution by placing v back on n. Our aspiration criteria is simple, should the best move chosen at a given iteration yield a solution with a cost lower than the best known solution throughout the Tabu search, then the virtual node and substrate node associated with that move will be freed from their Tabu status. Finally, we present our intensification and diversification strategies. We set two main diversification strategies : a "random restart" and "penalizing moves with high frequency". The random restart is launched in two occasions. First, in the case where the best candidate move was found to be a Tabu move that does not satisfy the aspiration criteria. Second, in the case where a Tabu iteration cannot find any candidate moves at k hops from the current node mapping solution. Here, any random move is eligible as long as it is not in the Tabu list, it satisfies the resource demand of the associated virtual node, and its move frequency is below a given threshold. This random restart allows to escape cases of local optimum, and leap towards the unexplored search space. The second diversification strategy consists of penalizing moves with a move frequency higher than a predetermined threshold, by multiplying the unit of move frequency with a constant c_3 (in our numerical results we set c_3 to 50) and adding it to the cost function. This also allows the Tabu search to explore new solutions by moving virtual nodes to physical nodes with a low move frequency.

The purpose of the intensification strategy is to intensify the search on a given solution that is only penalized for violating the differential-delay constraint. This may lead to a solution that eradicates this violation by finding longer paths to connect the source to the recipient nodes. The intensification strategy launches a new Tabu search, where the node mapping solution is fixed, and the move now consists of swapping the link mapping solution. A single move is concerned with swapping a single path between the source and a recipient node by another detour (possibly-longer) path. Here moves are evaluated using the same cost function, but without considering penalty $\tilde{P}_1$, and after each selected move, its associated recipient node and chosen detour path are marked as Tabu for the next x number of iterations. It could happen that the new detour path leads to a cycle in the tree. Here, a "cycle-breaker" link is chosen among the links in the cycle, and the paths in the multicast tree are adjusted accordingly. It is important to note that our Tabu-based search can be slightly modified to also accommodate unicast VNs, by simply neglecting loops.

For both our MVNE Tabu and the intensification Tabu search, the stopping condition is set to a predetermined number *numIter* of consecutive iterations with no improvement. The procedural details of our Tabu-based MVNE approach is illustrated in Algorithm 3.2.

Step 1 of the Tabu-search runs a greedy node embedding which mainly consists of sorting the list of substrate nodes and virtual nodes, subsequently it requires O($|N|log|N|$) and O($|V|log|V|$),respectively. Next, an initial link embedding solution is obtained by running Dijkstra to find the shortest path from the source to each recipient node, which is O($|T|.|N^2|$). Step 2 consists of running I Tabu search iterations, where each iteration $i \in I$ finds a set of candidate moves for each virtual node using BFS, which takes O($|V|.(|N| + |L|)$). Further, I' iterations of the intensification strategy may be performed, where each iteration $i' \in I'$ consists of running Dijkstra to find k shortest paths for each receiver $t \in T$, which takes O($k.|T|.|N^2|$). Hence, the worst-case runtime of the Tabu search algorithm is O($I.I'.k.|T|.|N|^2$), where I, I', and k are constants, hence it is O($c.|T|.|N|^2$).

3.6 Numerical Results

In this section, we assess how well our suggested heuristics perform compared to the MVNE-ILP over various substrate network topologies. Here, we evaluate the MVNE-Tabu, and the MVNE-H with its three different node embedding approaches; Namely, the MVNE-HNM, MVNE-HFF, and MVNE-HG. As we have previously mentioned, we solve the MVNE problem online upon the arrival of each VN request. Hence, we assume that the VN arrivals and departures follow a Poisson distribution. Our numerical results are mainly divided into two parts: performance evaluation and comparative analysis. For the performance evaluation, we compare the optimality gap and runtime of the three aforementioned MVNE-H variations and the MVNE-tabu against the MVNE-ILP. This allows us to evaluate the efficiency of our proposed approach. As for the comparative analysis, we compare the three MVNE-H variations and the MVNE-Tabu for various metrics and over multiple substrate networks. All our numerical evaluations are conducted using CPLEX version 12.4 on a pentium IV machine at 3.4GHz with 8 GB RAM.

3.6.1 Performance Evaluation

First, we evaluate the performance of the MVNE-Tabu, MVNE-HNM, MVNE-HFF, and the MVNE-HG against the optimal solution obtained by the MVNE-ILP model. To do so, we run the above methods over FatTree (k=4). We assume that the VNs arrival follows a Poisson

Algorithm 3.2 TabuSearch Algo($G^s, G^v, k, p, numIter$)

Step 1: Find an Initial Solution
M_N = *getInitial_ NodeMappingSolution()*;
if (M_N == null) **then** /**There are no feasible node mapping solutions for G^v*
return NULL
else
    M_E = *getInitial_ LinkMappingSolution()*;
    $M_{current} = M(M_N, M_L)$;
    $M_{current}$.cost = *computeCost();*
    $M_{best} = M_{current}$;
end if
Step 2: Begin Tabu-Search
while ($counter < numIter$) **do**
    $\hat{C} = \{\}$; /**Initialize set of candidate moves*
    for ($v \in V$) **do**
        $\hat{C}_v$ = *getCandidateMoves(v,p,k)*;
        $\hat{C} = \hat{C} \cup \hat{C}_v$
    end for
    if ($isEmpty(\hat{C})$) **then** $bestMove$ = *RndMove()*;
        $counter$++;
    else
        $bestMove$ = $getLowestCostMove(\hat{C})$;
    end if
    UpdateMoveFrequency(bestMove);
    if ($bestMove.TabuStatus == True$) **then** /**Check Aspiration Criteria*
        if ($bestMove.cost < M_{best}.cost$) **then** $counter = 0$;
            $bestMove$.TabuStatus = False;
            $M_{current}$ = *updateSol($M_{current}$, bestMove)*;
            $M_{current}$ = *runIntensification($M_{current}$)*;
            $M_{best} = M_{current}$;
        else
            $M_{current}$ = *updateSol($M_{current}$, RndMove())*;
            $counter$++;
        end if
    else
        $M_{current}$ = *updateSol($M_{current}$,bestMove)*;
        if ($bestMove.cost < M_{best}.Cost$) **then** $counter$++;
        else
            $M_{current}$ = *runIntensification($M_{current}$)*;
            $M_{best} = M_{current}$;
            $bestMove$.TabuStatus = True;
        end if
    end if
end while

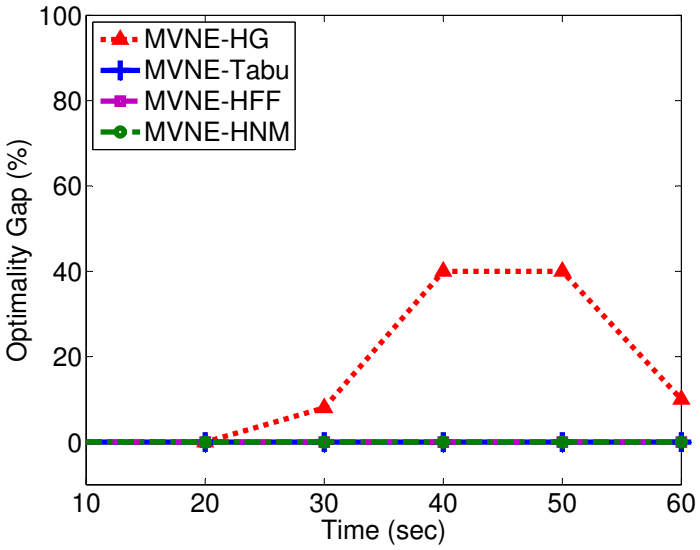


Figure 3.4: MVNE - Optimality Gap over FatTree ($k = 4$)

process distribution with a normalized load of 4, and the size of the VNs range between [3-12] receivers. To compare against the MVNE-ILP, we change the objective function of the MVNE-HNM to find the mapping solution with the lowest cost multicast tree, as depicted in Equation (3.30). Also, we assume that the substrate network has enough capacity to accommodate all the incoming VN requests.

$$Min \sum_{(i,j) \in L} z_{i,j}.b' \tag{3.30}$$

Over time, we measure the optimality gap between the optimal solution obtained by the MVNE-ILP, versus each one of the aforementioned heuristics. The results are illustrated in Figure 3.4. We observe that both the MVNE-HNM, MVNE-Tabu achieve an optimality gap of 0 when compared to the MVNE-ILP. Similarly, we observe that the obtained optimality gap of the MVNE-HFF is also 0. This is expected, since the First Fit node mapping technique aims to map the recipeint nodes within a given VN as close as possible to the root node; (later we show that as we increase the load in the network, MVNE-HFF is not always capable of finding feasible node mapping solutions that respect the end-delay and delay variation constraints, and thus renders a low admission rate). Finally, we observe that the MVNE-HG falls far from the optimal solution with an optimality gap that goes up to 40%.

Next, we look at the runtime (results are show in Table 3.1). We run each mapping algorithm for a single VN, and at every run we increase the number of recipient nodes. First, we observe that the runtime of the MVNE-ILP undergoes a steep increase as we augment

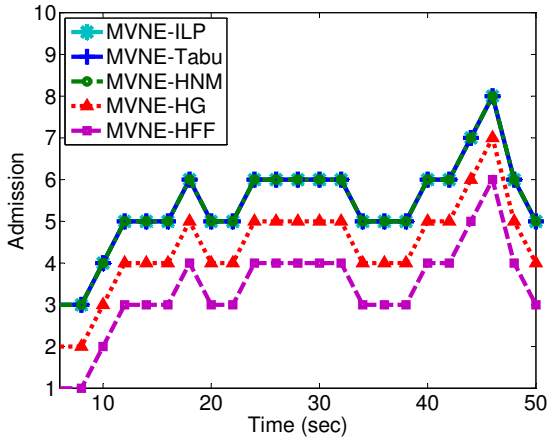
Table 3.1: MVNE - Runtime (ms)& Cost Evaluation for 1 VN Embedding over FatTree (k = 4)

# T	MVNE-ILP		MVNE-HNM		MVNE-HG		MVNE-HFF		MVNE-Tabu		Unicast VNE	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost
3	495	600	973	600	30	600	35	600	20	600	195	1000
4	595	1100	1393	1100	30	1100	39	1100	25	1100	314	2200
10	11613	2000	1901	2000	30	2000	35	2000	35	2000	440	5200
12	28592	2400	2616	2400	32	2400	35	2400	40	2400	525	6400

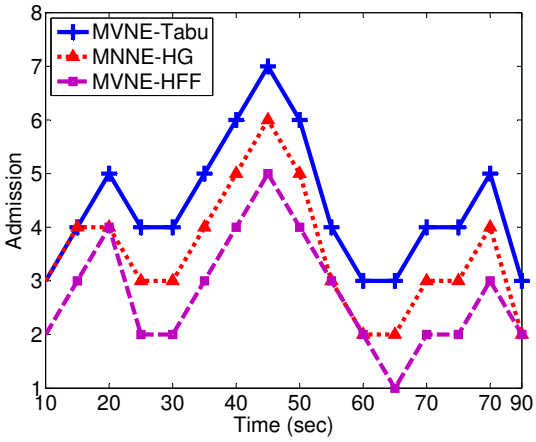
the size of the VN. On the other hand, the runtime of the MVNE-Tabu, MVNE-HG, and MVNE-HFF remains in the order of milliseconds, and that of the MVNE-HNM remains in the order of seconds. Clearly the MVNE-HG and the MVNE-HFF outperform the MVNE-HNM in terms of runtime, and this is mainly due to the fact that the MVNE node mapping model is an ILP.

To further convey the benefits of characterizing the type of communication in VNE, we measure the bandwidth cost of mapping the multicast VN as unicast versus multicast embedding. We observe that all the multicast embedding heuristics achieve the same bandwidth cost as the MVNE-ILP. However, the unicast embedding yields much more expensive embedding solutions. For a multicast VN with 3 recipient nodes, the unicast embedding renders a 40% more expensive embedding solution than any of the multicast embedding algorithms. This highly affirms our motivation that unicast embedding of multicast VNs severely impacts the network's utilization efficiency due to bandwidth wastage caused by redundant traffic.

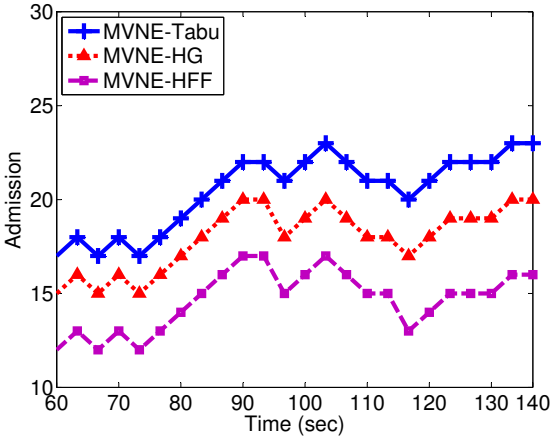
Moreover, we compare the runtime of our proposed heuristics (as shown in Table 3.2) over three networks with varying number of hosts (server racks) : FatTree (k=4) (#Hosts = 16, #Links = 32), a random network (#Hosts = 60, #Links = 90), and the FatTree (k = 8) (#Hosts = 128, #Links = 256). Here we randomly generated a set of 100 VNs of varying size [3-12] receivers over FatTree (k=4), and [3-30] receivers over FatTree (k=8) and the random network. We measure the average runtime of embedding 100 VNs using each of the aforementioned approaches. We observe again that the runtime of the MVNE-HFF and MVNE-HG is negligible. We also conclude that MVNE-Tabu is far more scalable than the MVNE-HNM, as it is capable of embedding VNs over the Fat Tree (k = 8) in less than a second, when the MVNE-HNM requires 13 minutes. This is mainly attributed to the fact that the MVNE-HNM adopts an ILP model to embed the nodes over multicast trees.



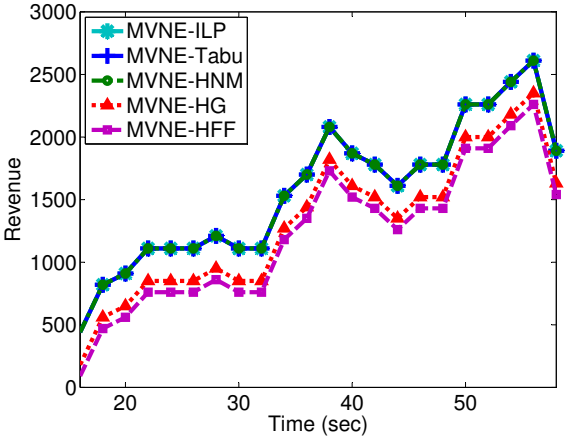
(a) Admission - FatTree $k = 4$



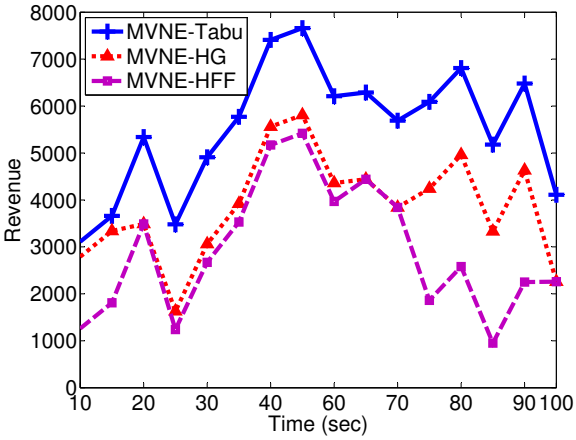
(b) Admission - FatTree $k = 8$



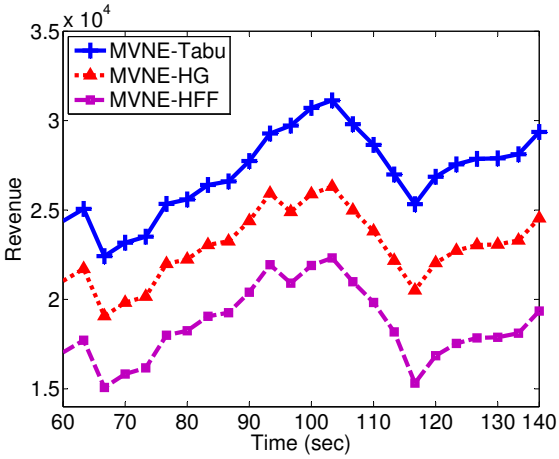
(c) Admission - Random($N = 60$,$L = 90$)



(d) Revenue - FatTree $k = 4$



(e) Revenue - FatTree $k = 8$



(f) Revenue - Random($N = 60$,$L = 90$)

Figure 3.5: MVNE - Comparative Analysis for Admission and Revenue

Table 3.2: MVNE - Average Runtime (ms) of Embedding 100 VNs over various substrates

	Heuristic			
	HFF	HG	Tabu	HNM
Fat Tee K=4	3	2	164	4309.06
Random	4	2	831	257830
Fat Tree K=8	5	4	870	780450
Fat Tree K=16	43,109	1,029	1,160	$> 18.10^5$

3.6.2 Comparative Analysis

In this section, we compare the MVNE-Tabu, MVNE-HNM, MVNE-HFF, and the MVNE-HG using various metrics; mainly we look at the blocking ratio and total revenue. Here, we use the load balancing objective function for the MVNE-HNM (as presented in Section 3.4), and we set the ρ value of the MVNE-HNM to 0.5, thus equally balancing the load on substrate nodes and links. We adopt three different substrate network topologies for our numerical analysis : FatTree (k=4) and (k =8) [14], since it is a commonly adopted topology for cloud data center. Further, we consider a random network with 60 substrate nodes and 90 substrate links. For all of the aforementioned substrates, we set the CPU node capacity of the servers in the FatTree network to 64 GB, and the substrate links's capacity to 10 Gbps. We use a randomly generated set of VN requests. Each VN request contains a source node, and a set of recipient nodes. The generated set varies for each considered substrate network, where we let the size of the VNs to be between [3-12] receivers when comparing over the FatTree (k=4), and [3-30] when comparing over the random and the FatTree (k=8) networks. However, in all the generated sets, we let the CPU demand of the virtual nodes is in the range [3-12] GB, while the bandwidth demand can vary between 50 and 500 Mbps. The end-delay threshold is a random number between [6-10] hops, while the differential-delay variation constraint ranges between [0-10] hops. These varying delay values allow to represent both delay-sensitive, and delay-agnostic multicast applications. We run each test case multiple times, and we present the average of 5 executions.

3.6.3 Admission Rate Over Time

The first metric we evaluate is the admission rate over time. Clearly a low admission rate indicates a poor quality mapping solution. We test this metric on the FatTree (k=4). We assume that the VNs arrival follows a Poisson distribution of load 4, and we look at the admission rate of each of the aforementioned embedding techniques over time. The results

are illustrated in Figure 5(a). From this Figure, we notice that the MVNE-Tabu and MVNE-HNM achieve the same admission rate attained by the MVNE-ILP. Whereas, the MVNE-HG and MVNE-HFF exhibit a fairly low admission rate due to their conceptual design that inhibits them from fully exploring the search space, particularly in the case of multicast VNs, where end-delay and delay-variations constraints are enforced. In fact, over FatTree (k=4), the MVNE-Tabu and MVNE-HNM achieve 12-33% higher admission rate than the MVNE-HG and 28-60% over MVNE-HFF. Similarly, we compare the admission rate over FatTree(k=8) and the random network. Here, we omit the MVNE-HNM from the test since it was found to be hard to scale. Again, we observe that the MVNE-Tabu outperforms the MVNE-HG and MVNE-HFF, within comparable embedding time (as shown in Table 3.2).

3.6.4 Revenue Over Time

The second metric we look at is revenue over time. Revenue is an important metric that complements the admission. A high admissibility does not necessarily indicate a higher revenue, since the algorithm could be admitting small VNs that do not generate much profit. The revenue for a given VN request is calculated using Equation (3.31), which consists of multiplying the sum of the CPU demands by a constant value that represents the profit per unit of CPU, and the sum of the bandwidth demand by another constant value that represents the profit per unit of bandwidth. Here, we put a higher weight on CPU, since CPU is a more expensive commodity in real-life [104], hence admitting VNs that require a higher amount of CPU demands are deemed to be more profitable. Throughout our test results, we set the ratio between the weights to 10.

$$Revenue = \prod_{cpu} \sum_{v \in \{s,T\}} c'_v + \prod_{bw} \sum_{e' \in E'} b'. \tag{3.31}$$

First, we look at the case of FatTree (k=4), and we measure the revenue over time for a poisson distribution with load factor 4. Again, we notice that the achievable revenue of the MVNE-Tabu and MVNE-HNM is equivalent to the revenue achieved by the MVNE-ILP over FatTree(k=4), and is significantly higher than its peers. Further, when comparing the MVNE-Tabu over the FatTree (k=8) and the random network, we also observe that the MVNE-Tabu outperforms the MVNE-HG and MVNE-HFF. This leads us to conclude that MVNE-Tabu is capable of admitting profitable VNs.

3.7 Conclusion

In this dissertation, we presented the multicast virtual network embedding for services with one-to-many communication. We aimed at solving this embedding problem in the frame of cloud computing data centers. We presented an optimal ILP formulation for solving the MVNE problem, which proved to be highly unscalable, due to its NP-Hard nature. In this regard, we presented a 3-steps MVNE mapping heuristic that employs a heuristic with an ILP model for solving the virtual node mapping subproblem. We evaluated our presented heuristic over various substrate network topologies, and against two widely used mapping heuristics: Greedy and First Fit. Our numerical results show that our MVNE model achieves a lower blocking ratio and a higher revenue.

Chapter 4

Towards Promoting Backup-Sharing in Survivable Virtual Network Design

4.1 Problem Statement

Failures in the physical infrastructure are common due to a multitude of reasons [47]. In fact, the year 2013 and 2014 have witnessed multiple cloud outages [105, 106], some of which got hold of major cloud providers (e.g., Amazon's EC2 cloud and Dropbox) causing millions of dollars in revenue loss. With millions of dollars at stake, attention converged towards solving the Survivable Virtual Network Embedding problem (SVNE) [48–55]. In addition, several studies [17,26,27] emerged to understand and characterize cloud computing hardware failure. Subsequently, based on real data traces from Microsoft data center, it has been found [26] that data center network hardware (routers, switches, and links) exhibit a high reliability of more than four 9s, where low commodity switches (ToRs and ASs) have a failure rate of less than 10%. Further, it was shown [26] that link failures tend to be isolated, with only 41% of link failures containing more than one link, and 10% consisting of more than 4 correlated link failures. Another study [17] have also looked at the pattern of network equipment failure, and have found that 50% of failures involved less than 4 devices, and 95% include less than 20 devices. This allows us to conclude that large correlated failures in cloud data center networks are rare. As for servers/facility nodes, the authors in [27] analyzed their failure characteristics using a real data center over the period of 14 months, and found an annual failure rate of 8%; with hard-disks being the predominant reason behind server failure. Further, the authors have also looked at successive failure rates, meaning the probability that a failed server would fail again after repair, and it was found that successive

failure rates are quite probable. For a 100 servers data center where 4 machines have failed more than once in 14 months, it was found that 20% of all repeat failures happen within a day of the first failure, while 50% happen within two weeks.

Given that isolated failures are more probable than correlated failures [26,27,58,107], most of the relevant literature considers the SVNE problem for single network component failure. In this chapter, we consider the case of single facility node failures. When a facility node fails, the hosted virtual node(s) needs to migrate to a backup facility node, as well as its associated connections to other virtual nodes belonging to the same tenant or virtual network. One way of achieving this failure recovery is by redesigning the VN request into a Survivable VN (SVN), and then mapping the resultant SVN onto the physical network. This redesign consists of augmenting the original VN with backup nodes. Each backup node is in charge of protecting one or many primary nodes. Hence, backup virtual links must be established between each backup node and the neighbors of the primary nodes it protects. Upon the failure of a facility node which hosts a virtual node v, v will migrate to its associated backup node, which will then resume the communication with v's neighbors. The augmented backup virtual nodes and links need to be provisioned with sufficient computing and bandwidth capacity to recover from any facility node failure. Given that a single facility node might fail at any point in time, the provisioned backup resources can be shared among the various backup nodes, since they will never be activated at the same time. Hence, backup resource sharing can be employed to minimize the backup footprint of the SVN upon embedding.

Designing a SVN encloses multiple challenges; among these challenges is the problem of deciding how many backup nodes to use and how to allocate these backup nodes to the primary nodes or virtual machines (VMs) in each VN such that we minimize the backup footprints in the substrate network. This problem is of paramount importance since these provisioned resources will remain idle until failures occur. Hence, over-provisioning can greatly impact the network's ability to admit future requests. Indeed, the cost-efficient SVN redesign problem has recurred multiple times in the literature [49, 53–55]. However, in all of the previous contributions, the number of backup nodes is fixed to either 1 or k (k being the number of critical nodes, nodes that demand a backup). Moreover, we observe that all of the existing redesign techniques are agnostic to the backup resource sharing in the substrate network, where this responsibility is delegated to the adopted mapping algorithm.

In this dissertation, we argue that fixing the number of backup nodes to either 1 or k could yield infeasible or even costly mapping solutions, and we provide several motivational examples to support this claim. In this regard, we introduce ProRed; a novel PROgnostic

REDesign approach that explores the space between 1 and k and promotes backup resource sharing in the substrate network. ProRed adopts a unique approach for the redesign; not only does it determine the augmented number of backup nodes and their connections to the primary nodes, but also their actual placement in the VN such that it minimizes the backup footprint at the substrate level. We provide solid theoretical foundations to prove that the location of the backup node in the VN network can positively impact the backup link routing in the substrate network, by increasing the achievable backup resource sharing. Hence, its prognostic property lays in its ability to promote the backup resource sharing prior to the embedding phase.

With the recent advances in cloud management tools and platforms (CMPs) (e.g. vRealize Suite [108], OpenStack [109]), failure contingency plans have become an integral module of CMPs, in order to facilitate disaster recovery and control in this heterogenous environment of the Cloud [110]. Our suggested redesign technique can be integrated within these recovery modules for a cost-efficient recovery against single facility node failure. Our numerical results prove that our suggested approach yields significant gain in terms of increasing the substrate network's admission rate, decreasing the amount of idle bandwidth in the substrate network, and boosting the overall revenue of the cloud provider.

4.2 Related Work

Survivability against facility node failure is of paramount importance, particularly in the case of critical services that do not tolerate failure. Indeed, this problem has attracted significant attention from the literature; here we can distinguish between single facility node failure [49], [53], [54], [55], and multiple facility nodes failure [52], [111], [112]. In the case of single facility node failure, the authors of [53] introduce a two-step approach to fully restore a VN from any single facility node failure. Mainly, their approach consists of augmenting the VN request with a 1-redundant or k-redundant backup nodes. The resultant SVN is then mapped onto the substrate network by placing virtual nodes in a given VN on distinct substrate nodes, while aiming to minimize the overall embedding cost. For this purpose, the authors introduce two backup-sharing techniques to minimize the incurred backup-bandwidth cost, namely cross-sharing and backup-sharing. The same problem is tackled in [49], here the authors consider the SVN to be given, and their aim is to map the SVN onto the substrate network while minimizing the amount of idle backup bandwidth. The virtual nodes

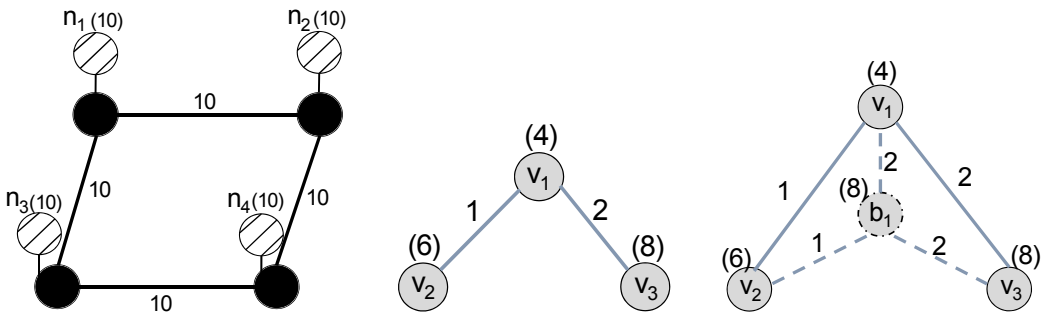
in a given VN can be mapped on the same substrate nodes, as long as their corresponding backup nodes are mapped on distinct nodes; this guarantees survivability against any single facility node failure. To embed the SVN onto the substrate network, two embedding heuristics are presented: A disjoint and a coordinated virtual node and virtual link mapping. For the disjoint embedding approach, a set of feasible node mapping solutions is first enumerated, then this set is passed on to an ILP model that picks the node mapping solution with the lowest reserved backup bandwidth, while the coordinated embedding adopts a link packing approach. Further, in [54], the authors present a novel approach for redesigning an SVN, denoted as Enhanced VN (EVN), and distinguish between failure-dependent and failure-independent EVN. The failure-independent EVN is similar to the 1-redundant SVN, while the failure-dependent EVN aims at minimizing the amount of idle backup resources by relaxing the constraint that only failed nodes will migrate. Instead, for each different failure-event, virtual nodes (primaries and backups) within a given VN will be re-arranged (migrated) differently to resume a working VN. Note that such approach incurs a considerable amount of migration overhead that can potentially cause a longer down-time. Moreover, in [55] the authors also adopt the 1-redundant SVN scheme to create an Auxiliary Protection Graph (APG). The APG is next embedded onto the substrate network using a Tabu-search meta-heuristic with cross-sharing and backup-sharing to minimize the backup footprints.

As for survivability against multiple facility node failure [52, 111, 112], the VN is augmented with the minimum number of backup nodes needed to guarantee a reliability degree r under a given probability of failure p. Further, in [52] and [112], the authors employ sharing across VNs in order to circumvent the inconvenience of idle resources. As for [111], the authors employ survivability at the inter-data center level, where a local protection approach is introduced to eliminate backup bandwidth over wide-area network.

Equal efforts have been devoted towards inaugurating effective protection schemes against substrate link failures [48], [113], [50], [114]. Here protection schemes can be mainly categorized as link-based and path-based protection. Further, few work in the literature tackled the case of correlated failure [51], [115], that is the case of single "regional" failure that brings down multiple substrate nodes and links at the same time. Substrate nodes and links that fail together are also referred to as a "shared risk group". Here risk groups are considered to be given and protection schemes are tailored for the case of a single risk group (regional) failure. A thorough taxonomy of the various failure scenarios and existing protection methods can be found in [116].

Our work is different as it tackles the current limitations of existing SVN redesign techniques, mainly the disadvantages associated with fixing the number of backup nodes, as well as their oblivion of the incurred backup footprint. ProRed circumvents these limitations by exploring the space between 1 and k, and adopting solid theoretical foundations to produce SVNs with backup-sharing properties.

4.3 Problem Definition



(a) Substrate Network (b) Virtual Network (c) Survivable Virtual Network

Figure 4.1: Substrate Network and Virtual Network Representation

1. **The Substrate Network :** We abstract the substrate (data center) network as an undirected graph denoted by $G^s = (N,L)$, where N is the set of substrate facility nodes, and L is the set of substrate links. Facility nodes are connected to the network via network nodes (routers/switches). Each substrate facility node $n \in N$ is associated with a finite computing capacity, denoted by c_n. Similarly, each substrate link $l \in L$ has a finite bandwidth capacity, denoted by d_l. Figure 1(a) illustrates a substrate network with 4 facility nodes, each with a CPU capacity of 10 units (represented by the number in parenthesis above each facility node). Similarly, we observe that the substrate links interconnecting the network nodes exhibit 10 units of bandwidth capacity (represented by the number on each substrate link).

2. **The Virtual Network (VN) :** A virtual network represents a client's request to deploy an application in a cloud data center. It consists of a set of virtual nodes (virtual machines), interconnected with virtual links. The virtual links correspond to the communication requirements between the virtual nodes in a given VN request. We denote a VN as a virtual graph $G^v = (V,E)$, where V represents the set of virtual nodes, each with a

CPU demand of c'_v, and e is the set of virtual links, each with a bandwidth demand of d_e. Every VN encompasses two type of VMs: critical and non-critical VMs; where critical VMs refer to the vital nodes that are imperative for the service's operability. Hence, such critical VMs do not tolerate any failure, and require backups; whereas non-critical VMs are tolerant to failures. All the VMs (critical or not) in a tenant's VN are denoted as primary VMs. Throughout this manuscript, we consider that all primary VMs in every VN are critical.

Figure 1(b) shows an example of a VN request with 3 virtual nodes interconnected via 2 virtual links, in addition to their associated CPU and bandwidth demands, respectively.

3. **Problem Definition :** Given the VN request, the SVNE problem aims to map this request onto the substrate network while providing survivability against single facility node failures. This can be done by redesigning the VN request into an SVN, which consists of augmenting the VN with backup nodes and provisioning enough bandwidth and CPU resources to recover from any facility node failure. The problem of designing survivable VNs encloses two major concerns: First, deciding how many backup nodes are needed to protect a given VN, and second, determining which backup node will be in charge of protecting which set of critical nodes. These two concerns highly depend on the substrate network capacity. On one hand, provisioning a high number of backup nodes and links greatly decreases the substrate network's admission rate, since these resources will remain idle until a failure occurs. On the other hand, limiting the number of backup nodes to a pre-determined constant may yield infeasible mapping solutions. Hence, finding the optimal design of reliable VNs consists of finding the tradeoff between the amount of backup resources provisioned and the efficient utilization of the substrate network. The SVN redesign problem can thus be formulated as follows:

   Problem Definition 4.1. *Given a substrate network $G^s = (N, L)$, and a VN request G^v = (V,E), Find the optimal redesign D of the given VN request G^v into a survivable VN, such that the amount of backup idle resources in the substrate network is minimized, while guaranteeing survivability against single facility node failure.*

One way to solve the problem is by enumerating all possible designs $d \in D$, where each d can contain between 1 to k backup nodes. For a given number of backup nodes i ($2 \leq i \leq k$), there could exist multiple designs d. These designs are represented by the different ways the V virtual nodes are divided into i clusters, where each cluster is protected by a single backup node. This is similar to the various ways o distinct objects can be distributed into m different

bins with k_1 objects in the first bin, k_2 in the second, etc. and $k_1+k_2+....k_m = o$. This indeed is obtained by applying the multinomial theorem where $\sum_{k_1+k_2+...+k_m=o} \binom{o}{k_1+k_2+...+k_m} = m^o$. Therefore, for V virtual nodes and i backup nodes, there are $|V|^i$ different mapping designs. Once the set of all possible designs d is enumerated, it can be fed to an Integer Linear Programming (ILP) model to determine the optimal design d that achieves the lowest amount of backup idle resources in the substrate network. It is important to note that in order for the model to determine the optimal design, it requires to solve the SVNE for each design d; this renders the problem NP-Hard.

In this regard, we reformulate the problem to seek a redesign approach that promotes backup sharing in the substrate network, hence it is inheritably capable of minimizing the backup footprints. In section 4.5, we introduce a heuristic-based redesign approach that renders such prognostic SVNs.

4.4 The SVN Redesign Problem

4.4.1 Limitations of Conventional VN Redesign Techniques

One of the most commonly adopted redesign techniques for recovery against single node failures are formally known as the 1-redundant and k-redundant schemes. In the case of the 1-redundant scheme, the VN request is augmented with a single backup node that needs to be connected to the neighbors of each critical node via backup virtual links. Next, the resultant SVN is embedded onto the substrate network while forcing the primary and backup nodes in a given SVN to occupy distinct substrate nodes. This ensures that a single substrate node failure will not affect more than one virtual node in the same VN request. Figure 1(c) illustrates the case where the VN request presented in Figure 1(b) is augmented with a single backup node b_1, as per the 1-redundant scheme. The backup node must be provisioned with the maximum CPU demand of all the critical nodes, so it can assume any single facility node failure. Hence 8 units of CPU is reserved on backup node b_1. Moreover, for each backup virtual link connecting b_1 to any critical node v, it is sufficient to reserve the maximum bandwidth demand on v's adjacent links, since backup link (b_1,v) will only be activated upon the failure of one of v's neighbors. For example, the backup link (b_1,v_1) will only be activated in the case where virtual node v_2 or v_3 fails. In the case where v_2 fails, 1 unit of bandwidth is required to resume the communication on backup link (b_1,v_1). Similarly, in the case where v_3 fails, it will also migrate to b_1 and communicate with v_1 with 2 units of bandwidth. Given that at any point in time either v_2 or v_3 would fail, it is sufficient

to reserve 2 (rather than 3) units of bandwidth on the link connecting b_1 to v_1. The set of backup links that are activated simultaneously upon the failure of a virtual node v is denoted as the Backup-Group of v ($BG(v)$) [53]. For instance, the $BG(v_1)$ contains backup links (b_1,v_2) and (b_1,v_3). Similarly, the backup group of $BG(v_2)$ and $BG(v_3)$ is (b_1,v_1).
Now, for the k-redundant scheme, the VN is augmented with k backup nodes, where k represents the number of critical virtual machines (or nodes). In this case, each backup virtual node protects a single primary node, and hence it only connects to its neighbors via backup virtual links. Each backup node along with its associated backup links will be provisioned with the same amount of resources as the primary node it protects and its adjacent links, respectively. When a facility node fails, only the affected node will be

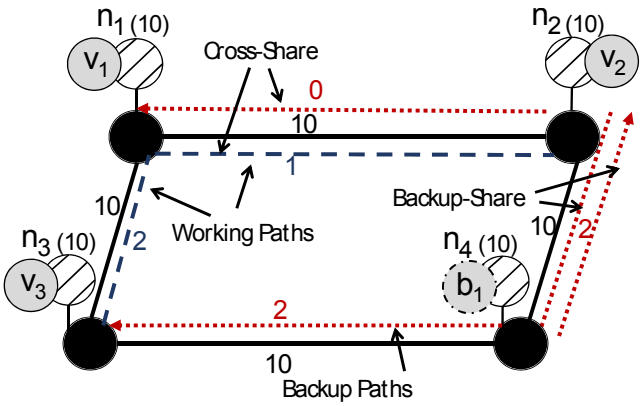


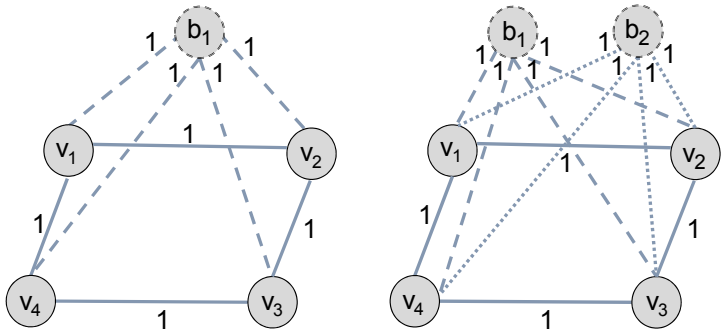
Figure 4.2: Backup Resource Sharing

disconnected from the substrate network. However, its adjacent network node and substrate links will remain active and capable of routing traffic. Thus, upon the failure of a facility node that hosts a virtual node v, the bandwidth on the original working paths that connect v to its neighbors in the substrate network will be released, and hence becomes available. This released bandwidth can thus be reused by the corresponding backup paths of v's backup node. Such type of sharing is known as $cross - sharing$ [53] between working and backup paths. Each virtual node v is associated with a Working-Group ($WG(v)$) that contains the set of v's working paths. For instance, the $WG(v_1)$ contains (v_1,v_2) and (v_1,v_3). Hence, the $BG(v_1)$ can reuse the bandwidth of the $WG(v_1)$ upon v_1's failure through cross-sharing. Moreover, given that a single node might fail at any point in time, the backup paths belonging to different backup groups can share their bandwidth in the substrate network. Such type of sharing is referred to as $backup - sharing$ [53]. Figure 4.2 shows a mapping solution for the 1-redundant SVN presented in Figure 1(c) over the substrate network in Figure 1(a). We observe that for backup link (b_1,v_3), 2 units of bandwidth need to be reserved, since

the substrate links which route this backup path do not overlap with any other appropriate backup or working paths. However, backup paths (b_1,v_1) and (b_1,v_2) overlap over substrate link $\{n_2,n_4\}$; and given that these backup paths belong to distinct backup groups, only 2 units of bandwidth need to be reserved on substrate link $\{n_2,n_4\}$, rather than 3 due to backup-sharing. Moreover, backup path (b_1,v_1) further overlaps with working path (v_1,v_2) on substrate link $\{n_1,n_2\}$; hence 0 unit of bandwidth needs to be reserved on this substrate link via cross-sharing.

The problem with the 1-redundant and k-redundant schemes is that by forcing the number of backup nodes to be either 1 or k, we may end-up with infeasible or costly mapping solutions. This is due to the fact that the substrate might not have enough bandwidth capacity to route the traffic between 1 backup node to the neighbors of all critical nodes, in the case of the 1-redundant scheme. Whereas, in the case of the k-redundant scheme, a substantial amount of CPU resources remain idle until a failure occurs, since k-redundant requires as many backup nodes as primary critical nodes, not to mention the large number of backup virtual links needed to associate each backup node with its appropriate primary critical node. This motivates the need for a cost-efficient redesign technique that is capable of exploring the space between 1 and k, and finding the balance between the amount of provisioned CPU and bandwidth to yield feasible and cost-efficient embedding solutions.

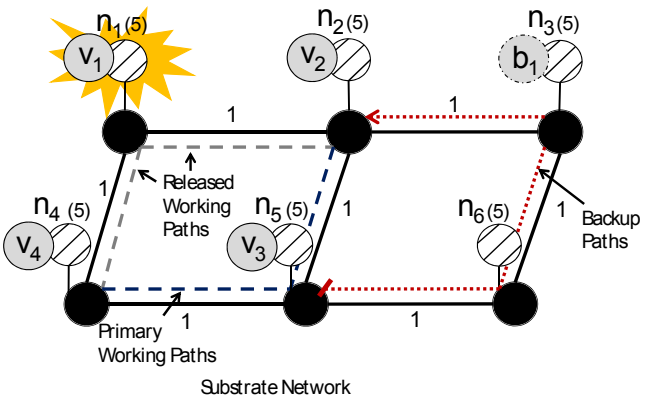
4.4.2 Illustrative Example



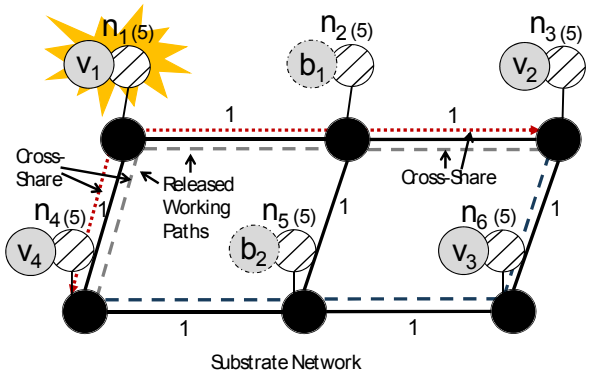
(a) 1-Redundant SVN (b) 2-Redundant SVN

Figure 4.3: Survivable Virtual Network Design Schemes

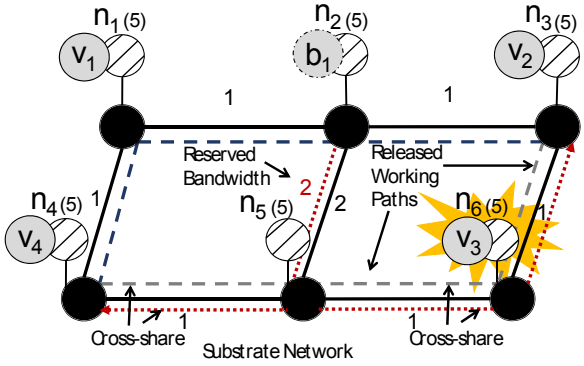
To further illustrate the inconvenience of the conventional redesign techniques, consider the case of a 4 nodes VN, where each virtual node is considered to be critical. Using the 1-redundant scheme, we augment this VN with a single backup node, connected to the neighbors of all critical nodes via backup virtual links, as illustrated in Figure 3(a). Now,



(a) 1-Redundant SVN Embedding Solution (1)



(b) 2-Redundant SVN Embedding



(c) 1-Redundant SVN Embedding Solution (2)

Figure 4.4: Designing and Embedding Reliable VNs

consider a substrate network with 6 facility nodes interconnected via substrate links, each with a bandwidth capacity of 1 unit, as shown in Figure 4(a). Given the 1-redundant SVN, there exists no feasible mapping solutions on the aforementioned substrate network. For instance, consider embedding the SVN using the mapping solution illustrated in Figure 3(a). When substrate node n_1 fails, virtual node v_1 migrates to b_1 which needs to communicate with virtual nodes v_2 and v_4. b_1 is capable of reaching virtual node v_2 through path $\{n_3\text{-}n_2\}$. However, the substrate network's capacity, with the current embedding solution inhibits b_1 from reaching node v_4, since the working path of $\{v_3\text{-}v_4\}$ remains operational, occupying the 1-unit of bandwidth on the substrate link $\{n_4\text{-}n_5\}$. This renders the embedding solution illustrated in Figure 4(a) infeasible. By examining all possible mapping solutions of the 1-redundant SVN on the given substrate network, we find that they are all infeasible. This is because the 1-redundant scheme connects a single backup node to the neighbors of all critical nodes. Hence b_1's bandwidth demand along with the given substrate network capacity, inhibits b_1 from protecting this VN against any single node failures.

On the other hand, consider the case where the aforementioned VN is augmented with 2 backup nodes b_1 and b_2, as shown in Figure 3(b). b_1 assumes the failure of critical nodes v_1 and v_2, and b_2 replaces v_3 and v_4 in case any of them failed. Upon embedding the resultant SVN, we notice that this survivable design does indeed yield a feasible solution and requires 0 unit of reserved bandwidth due to cross-sharing, as illustrated in Figure 4(b). For example, consider the case where the facility node n_1 fails; subsequently, v_1 will migrate to b_1, and that latter needs to resume v_1's communication with v_2 and v_4. The failure of virtual node v_1 leads to the release of the active bandwidth on working paths $\{n_1\text{-}n_2\text{-}n_3\}$ and $\{n_1\text{-}n_4\}$ connecting virtual node v_1 to v_2 and v_4, respectively. The released bandwidth will be reused by b_1 to reach v_2 and v_4 through cross-sharing. By employing cross-sharing for all other virtual node failures in the given VN, we can conclude that indeed the 2-redundant SVN requires 0 unit of reserved bandwidth.

Further, consider the same substrate network, where link $\{n_2\text{-}n_5\}$ has a capacity of 2 units, as illustrated in Figure 4(c). In this case, we can indeed find a feasible embedding solution for the 1-redundant SVN with a provisioned bandwidth cost of 2 units, whereas the 2-redundant scheme still requires 0 unit of provisioned bandwidth.

These motivational examples support our claim that by forcing the number of backup nodes to be either 1 or k, we might end up with infeasible or costly mapping solutions. Whereas when we augment the VN with i ($1 \leq i \leq k$) backup nodes ($i = 2$ in the above example), we achieve a balance between the amount of backup bandwidth and CPU that needs to be reserved. In fact, this balance yields a feasible solution, when the 1-redundant and k-redundant fail to find one.

This motivates the need for a redesign approach that is capable of finding that balance, rather than being fixed to either 1 or k backup nodes. By exploring the space in the range between 1 and k, we can obtain lower-cost mapping solutions, and increase the network's admissibility. This is one of ProRed's unique capabilities. Another advantage of ProRed is that it redesigns the VN in a way to promote the backup bandwidth sharing at the substrate network. In the next section we present ProRed's theoretical foundation that enables it to fulfil these two promises.

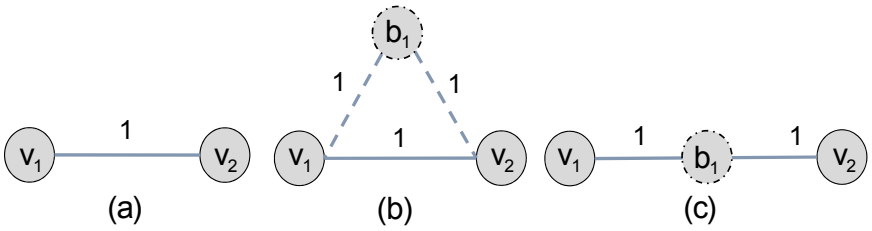


Figure 4.5: Theoretical Foundation

4.5 Prognostic Redesign Approach (ProRed) :

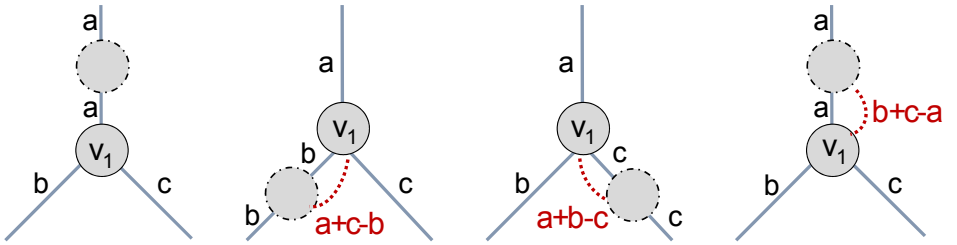
4.5.1 Theoretical Foundation

In this section, we present the theoretical foundation on which ProRed's redesign technique is established. We begin our explanation with a motivational example: Consider a 2 nodes VN illustrated in Figure 4.5(a). Augmenting the VN with a single backup node, using the 1-redundant scheme, requires 2 units of backup bandwidth (as shown in Figure 4.5(b)). By employing an effective embedding approach, this estimated bandwidth cost could be minimized at the substrate network level via cross-sharing and backup-sharing. Observe, however, that by placing this backup node along the path connecting v_1 and v_2, the resulting SVN will require the least amount of backup bandwidth to-be provisioned upon embedding. This is due to the fact that by placing the backup node in between its associated primary nodes, we encourage the backup path to be routed through the primary path connecting v_1 and v_2 in the substrate network. Subsequently, if either one of these primary nodes fail, the backup node will cross-share (reuse) the released primary bandwidth. It should be noted here that this redesign approach is indeed prognostic to backup resource sharing, as it is able to predict (promote) the cross-sharing (bandwidth reuse) at the VN level. Indeed, throughout our numerical results, we show that ProRed achieves considerable gains in terms of reducing the total bandwidth cost against the conventional redesign techniques, and greatly decreasing the network's blocking ratio.

We build on this motivational example to formulate a novel redesign technique that determines the location of backup nodes in the VN, such that cross-sharing and backup-sharing can be fully promoted in the substrate network. Incrementing a backup node for every two virtual nodes is definitely costly in terms of idle CPU resources. Hence, we resort to clustering a subset of virtual nodes into distinct sets, where nodes in a particular set are covered by a single backup node. In each set, the backup node is positioned such that the maximum amount of backup resource sharing can be achieved upon the embedding. This clustering

technique can thus create a balance between the amount of provisioned backup nodes and links.

To create a set, we begin by selecting the virtual node with the highest degree. This allows a larger number of primary virtual nodes to be clustered within a single set, which can substantially decrease the amount of reserved CPU resources. Once the starting node is identified, we place the backup node on the adjacent link with the highest bandwidth demand, which can yield the most backup resource sharing. To support this claim, consider



(a) Total Cost = 2a + b + c (b) Total Cost = 2a + b + 2c (c) Total Cost = 2a + 2b + c (d) Total Cost = a + 2b + 2c

Figure 4.6: Designing Reliable VNs

the following example illustrated in Figure 4.6. Let v_1 be the node with the highest nodal degree 3. Its adjacent links have a bandwidth demand of a, b and c, respectively. We assume (without loss of generality) :

$$a > b > c \tag{4.1}$$

In order to protect v_1, we need to place a backup node on one of its adjacent links. In this case, we have 3 different links to choose from, we can either place the backup node on the link with bandwidth demand a, b, or c. These three different options are illustrated in Figure 6(a), 6(b), and 6(c), respectively. Now, we distinguish between two scenarios:

- Case 1: a $\geq$ b+c

  By placing the backup node on the link with the highest bandwidth demand a (shown in Figure 6(a)), and given that a $\geq$ b+c, this implies that 0 unit of bandwidth is required on this *virtual backup link. This is due to the fact that the backup node is able to reach all of v_1's neighbors by reusing the primary bandwidth through cross-sharing.* Whereas, by placing the backup node on the link with bandwidth demand b (shown in Figure 6(b)) additional bandwidth needs to be reserved in order to reach v_1's neighbors. In fact, since $b < a$, the backup node cannot reach v_1's neighbor at link a without reserving an additional (a - b) units of bandwidth. The same applies to reach v_1's neighbor at link c, hence an overall (a + c - b) units of bandwidth needs

to be reserved. This renders a total cost of $(2a + b + 2c)$, which is obviously more expensive than the redesign solution presented in Figure 6(a).

- Case 2: a < b+c

  Now in the case where $a \leq (b + c)$, placing the backup node on link a implies that a total of $(b + c - a)$ must be reserved on the link connecting the backup node and v_1, as illustrated in Figure 6(d). However, this solution still renders the lowest total cost than that achieved by placing the backup on any other link.

Overall, we can conclude that in case 1 or case 2, placing the backup on the link with the highest bandwidth will always yield the lowest total cost.

4.5.2 ProRed Algorithm :

Algorithm 4.1 ProRed: Prognostic Redesign Heuristic

1: **Given** $G^v = (V, E)$ */*Virtual Network Topology*/*
2: */*Set cover flag for nodes and links to false*/*
3: **for** $(v \in V)$ **do**
4: $v.covered$ = false;
5: **end for**
6: $C = \{\}$; */*Initialize the list of covered nodes*/*
7: **while** $(|C| < |V|)$ **do**
8: $\hat{C} = \{V\}$ - C;
9: **Step 1: Find Starting Node**
10: $v_1 = GetNodeWithMaxNodalDgr(\hat{C})$;
11: $\tilde{E} = GetAdjacentLinks(v_1, \hat{C})$;
12: **Step 2: Find Starting Link**
13: $e = GetLinkWithMaxBW(\tilde{E})$;
14: **Step 3: Create a new Set**
15: $s = CreateSet(v_1, e)$;
16: $C = C \cup s$;
17: **end while**

In this section, we present the SVN redesign heuristic that is founded on the theories and observations presented in Section 4.5.1. The objective of this algorithm is to assign a backup node for each critical node in the given VN topology; we refer to a critical node that is assigned to a backup node as *covered* (or protected). Algorithm 4.1 presents the 3 steps approach of our ProRed algorithm. Initially, all the virtual nodes in the VN topology are considered as *uncovered*; hence, we initialize the virtual nodes with a cover flag set to false.

Next, we define two new sets C and $\hat{C}$ that are updated at the end of every iteration with the list of covered and uncovered nodes, respectively. The process terminates when C contains all the critical nodes in the VN request. At each iteration, the algorithm creates a single set. We define a set as an ensemble of critical nodes protected by a single backup node. To create a set, we first need to identify a starting point, from which a set will begin and grow. Based on the previous observations presented in Section 4.5.1, the starting point is defined by node v_1 with the highest nodal degree in the list of uncovered nodes $\hat{C}$ (Line 10), where the nodal degree count only considers v_1's neighbors in $\hat{C}$. Next, we need to find v_1's adjacent link with the highest bandwidth demand. To do so, we need to find the set $\tilde{E}$ of v_1's adjacent links with both nodes in $\hat{C}$ (Line 11), and then pick the link e with the highest bandwidth (Line 13). Finally, the algorithm invokes the *CreateSet* function that returns a set s which contains the critical nodes covered by the newly discovered set. The nodes covered by set s will thus be added to set C (Line 16) in order to prevent selecting these nodes as starting points for new sets in future iterations. In Algorithm 4.2, we highlight the procedural details of the *CreateSet* function. It begins by creating a new backup node b to be placed between the edge nodes v_1 and v_2 of link e. To exploit cross-sharing, virtual link e is replaced by two backup virtual links $\hat{e}_1$ and $\hat{e}_2$ that position backup node b in between nodes v_1 and v_2. This would encourage the backup virtual link to be routed through the primary path connecting nodes v_1 and v_2, thereby promoting cross-sharing upon embedding.

Initially, both virtual links $\hat{e}_1$ and $\hat{e}_2$ will be provisioned with primary bandwidth demand of link e, denoted as d_e, and the CPU demand of b is set to the maximum CPU demand of critical nodes v_1 and v_2 (line 6). The backup bandwidth to be provisioned on link $\hat{e}_1$, denoted as $d_{\hat{e}_1}$, is set to be the sum of the bandwidth demands of v_1's adjacent links (excluding the link connecting v_1 to v_2) minus the primary bandwidth demand d_e of primary virtual link e (Line 8), only if the sum is greater than d_e, since this suggests that the released bandwidth on e_1 is not sufficient to recover the communication between v_1 and its adjacent nodes upon the failure of v_1 (Line 7). It is important to note that all of v_1's adjacent links (both in C and $\hat{C}$) are considered in the computation of $d_{\hat{e}_1}$, since we need to provision enough bandwidth on $d_{\hat{e}_1}$ to allow b to reach all of v_1's neighbors upon failure. The same applies when assigning the reserved bandwidth on link $\hat{e}_2$, denoted as $d_{\hat{e}_2}$ (Line 8). Subsequently, nodes v_1 and v_2 are now protected (covered) by backup node b. Once this set is established, we need to grow it in order to cover the highest number of adjacent nodes possible without incurring too much additional backup bandwidth. First, we need to include all the adjacent leaf nodes in the set, otherwise leaf nodes will be left uncovered, or would require a dedicated backup node, which

Algorithm 4.2 CreateSet(virtual_node v_1,virtual_link e)

1: $s = \{\}$;
2: $v_2 = GetOtherNode(v_1,e)$;
3: **Step 4: Create a new backup node** b
4: $\hat{e_1}$ = new virtual_link(v_1,b,d_e);
5: $\hat{e_2}$ = new virtual_link(v_2,b,d_e);
6: $setCPU(b,\max(v_1,v_2))$;
7: **if** (Sum($GetAdjacentLinksBandwidth(v_1)$) $\geq d_e$) **then**
8: $d_{\hat{e_1}}$ = Sum($GetAdjacentLinksBandwidth(v_1)$) - d_e;
9: **end if**
10: **if** (Sum($GetAdjacentLinksBandwidth(v_2)$) $\geq d_e$) **then**
11: $d_{\hat{e_2}}$ = Sum($GetAdjacentLinksBandwidth(v_2)$) - d_e;
12: **end if**
13: $v_1.covered = v_2.covered$ = true;
14: $s = s \cup \{v_1,v_2\}$;
15: **Step 5: Protect Adjacent Leaf Nodes**
16: $T = v_1.getAdjacentLeafNodes()$;
17: **while** (T.hasNext()) **do**
18: $t = T$.next();
19: $t.covered$ = true;
20: $s = s \cup \{$t$\}$;
21: $setBW(\hat{e_1},\max(d_{\hat{e_1}},d_{(v_1,t)}))$;
22: $setCPU(b,\max(b,t))$;
23: **end while**
24: Repeat Step 5 Lines (16-23) for v_2
25: **Step 6: Protect Adjacent non-leaf Nodes**
26: $R = v_1.getAdjacentNonLeafNodes()$;
27: **while** (R.hasNext()) **do**
28: $r = R$.next();
29: **if** ($(2d_{(v_1,r)} \geq$ Sum($GetAdjacentLinksBW(r)$))&&
30: ($d_{\hat{e_1}} \geq d_{(v_1,r)}$ && ($r.hasAdjacentLeafNodes()$ = null)) **then**
31: $r.covered$ = true;
32: $s = s \cup \{r\}$;
33: $setCPU(b,\max(b,t))$;
34: **end if**
35: **end while**
36: Repeat Step 6 (Lines 26-35) for v_2
37: **return** s;

is seemingly not cost efficient. To cover leaf nodes, we need to adjust the bandwidth demand on links $\hat{e}_1$ and $\hat{e}_2$, appropriately, with enough bandwidth to assume the failure of any leaf node, as well as the CPU demand of backup node b. Finally, the algorithm will also attempt to cover non-leaf neighbor nodes of v_1 and v_2 using backup-sharing, meaning without assigning any additional bandwidth on backup virtual links $\hat{e}_1$ and $\hat{e}_2$, thereby getting a "free-ride" by joining the current set. Thus, adding non-leaf nodes to the current set can ultimately reduce the amount of provisioned backup resources needed to protect the virtual network. Given a non-leaf neighbor node v' of v_1, if the sum of the bandwidth demand on v''s adjacent links including link (v',v_1) is smaller than the reserved bandwidth on link $\hat{e}_1$, and excluding the bandwidth demand on link (v',v_1), smaller than the bandwidth demand on link (v',v_1); further, if (v',v_1) is the link with the highest bandwidth demand among v''s adjacent links, then v' could be included in v_1's set and subsequently protected by backup node b without incurring any additional backup bandwidth via backup-sharing. Finally, the algorithm returns the set of nodes that are covered by the newly created set s. The *CreateSet* function has a complexity of O($|V|$), which renders the complexity of ProRed's redesign heuristic to be O($|V|^2$), since we call the *CreateSet* function for each uncovered node in the VN request. This set-based protection scheme enables ProRed to assume multiple facility node failures, in the event where these failures affect VMs in distinct sets (as opposed to 1-redundant and k-redundant schemes that only support single node failure). This property can be leveraged to enhance the fault-tolerance of VNs by spreading the VMs in a single set over multiple fault-domains (e.g., a Top-of-Rack switch in a FatTree [14] topology).

It is important to note that the actual amount of reserved backup bandwidth in the substrate network depends on the quality of the adopted SVNE approach. It can indeed be substantially reduced with a highly-efficient embedding approach that exploits cross-sharing and back-sharing; or it can get aggravated if the backup node was poorly placed far from the primary virtual nodes, requiring multiple hops to reach them. In the case of ProRed, the placement of the backup node at the VN level guarantees the predicted cross-sharing that the resultant SVN will enjoy once embedded onto the substrate network.

4.5.3 Illustrative Example

To further illustrate the enactment of ProRed's redesign algorithm, consider the VN topology presented in Figure 7(a). The algorithm begins by identifying a starting node and link, which in this case are node v_7 with link $\{v_4,v_7\}$, since they correspond to the node with the highest degree, and its adjacent link with the highest bandwidth demand. Next, a set is created by

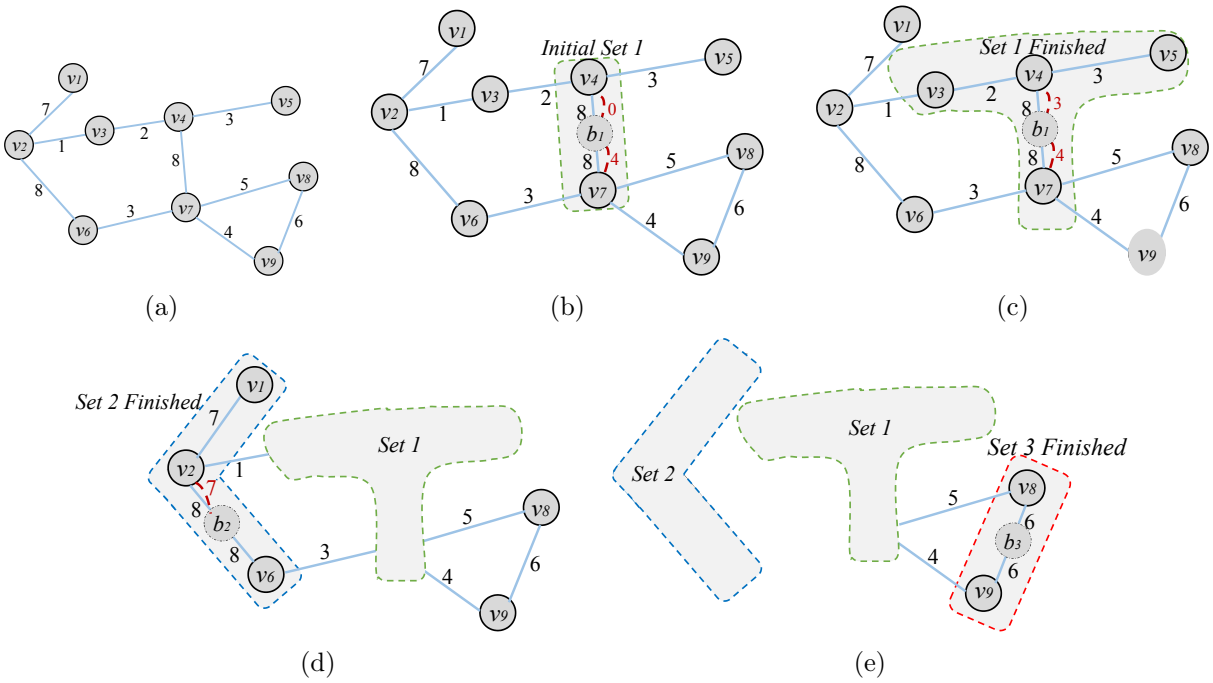


Figure 4.7: Step-by-Step SVN Redesign Algorithm.

placing a backup node b_1 on link $\{v_4,v_7\}$, as shown in Figure 7(b). This implies that nodes v_4 and v_7 are now protected by backup node b_1. Since the sum of the adjacent links to v_4 (excluding link $\{v_4,b_1\}$) is smaller than the bandwidth on link $\{v_4,b_1\}$, 0 unit of bandwidth is required to protect node v_4. When v_4 fails, the bandwidth reserved on the substrate paths routing virtual links $\{v_4,v_7\}$, $\{v_4,v_3\}$, and $\{v_4,v_5\}$ will be released. Now, v_4 will migrate to b_1 and that latter needs to resume v_4's communication with v_3, v_5 and v_7; b_1 will thus reuse 8 units of released bandwidth on the path connecting v_4 and v_7 to reach v_7. Similarly, b_1 will reuse 2 units of the released bandwidth on the path connecting $\{v_4,v_7\}$ and $\{v_4,v_3\}$ to reach v_3, and 3 units on the substrate paths connecting $\{v_4,v_7\}$ and $\{v_4,v_5\}$ to reach v_5.

Now to protect virtual nodes v_7, we observe that the sum of its adjacent links is 12, which implies that 4 additional units of bandwidth must be reserved on link $\{v_7,b_1\}$ in order to protect node v_7. This is because when v_7 fails it migrates to b_1, that latter now needs to go through the path connecting b_1 to v_7 and then cross-share the released bandwidth on the paths connecting v_7 to v_6, v_8 and v_9. Now, given that only 8 units of bandwidth is released on $\{v_7,b_1\}$; hence, 4 additional units must be reserved to fully protect v_7.

Next, the set is grown by adding the adjacent leaf nodes of v_4 and v_7. The only leaf node found is v_5 which will be added to the set, and subsequently incurs 3 units of backup bandwidth to be reserved on link $\{v_4,b_1\}$. Finally, the potential of adding non-leaf nodes is explored.

Indeed, we find that only node v_3 can be added to the set with no additional bandwidth, as shown in Figure 7(c). When no additional nodes can be further added to the set, the set becomes saturated. Subsequently, the *CreateSet* function returns set s_1 with backup node b_1 protecting virtual nodes v_3, v_4, v_5, and v_7, which leaves 5 critical nodes in the given VN uncovered. Hence, a new set is initiated starting with node v_2, since it represents the next uncovered node with the highest nodal degree. The same process repeats, and returns set s_2 with backup node b_2 protecting virtual nodes v_1, v_2, and v_6, as shown in Figure 7(d). Finally, set s_3 is created with backup node b_3 covering nodes v_8 and v_9, illustrated in Figure 7(e). Once all critical nodes are protected, the algorithm terminates. At the end, we obtain 3 sets with 3 backup nodes protecting 9 critical nodes and an estimated 14 units of backup bandwidth to be reserved. However, in the case of the 1-redundant scheme, a single backup node b needs to connect to each virtual node; hence a total of 9 backup links are needed. This means that potentially 56 units (sum of backup bandwidth to be provisioned) of bandwidth needs to be reserved to connect b to the virtual nodes.

4.6 The SVN Embedding

Upon obtaining the redesigned VN, the next step is to embed the latter onto the substrate network. Since the SVNE problem is NP-Hard, we adopt a disjoint mapping approach, where we perform the node mapping first and then the link mapping. For the node mapping, we use the VMP algorithm in [49] to find a set $\mathcal{M}$ of feasible node mapping solutions. Note both the primary and backup node placement is performed jointly. Here, we sort the substrate nodes in order of proximity (e.g. pods in a FatTree), and the VMs placement is performed set by set. This encourages the backup node in each set to be placed on substrate nodes within close proximity of the primary VMs it protects, thereby attempting to replicate as-much-as-possible the virtual design performed by ProRed. Next, the link embedding is performed to route the primary and the backup links. At the embedding phase, we separate the primary virtual links from the augmented backup links. That is, going back to our example presented in Figure 7(b), backup links (b_1-v_4) and (b_1-v_7) are routed separately from the primary link (v_4-v_7); and the predicted backup resource sharing will be realized on all the common links traversed by the primary and the backup paths. This prevents the primary path from being routed through the backup node, particularly when performed in substrate topologies where facility nodes are interconnected by multiple layers of network nodes.

4.6.1 The SVN Embedding Model (SVNE-M)

For the link mapping, first we formulate an ILP model (denoted as SVNE-M) that performs the primary and backup links embedding jointly. The SVNE-M will then select the lowest cost mapping solution $m \in \mathcal{M}$, and determine its corresponding link mapping solution. The SVNE-M can thus be formulated as follows:

- Parameters:

$G^s = (N, L)$: substrate network with N nodes and L links.
G^v=(V, E) : virtual network with V virtual nodes and E virtual links.
$\hat{E}$: the set of backup virtual links.
$\mathcal{M}$: the set of all node mapping solutions.
S : the list of constructed sets.

$$\delta_{v,n}^{m} = \begin{cases} 1, & \text{if } v \text{ is mapped onto substrate node } n \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$$

- Decision Variables:

$$x_m = \begin{cases} 1, & \text{if node mapping solution } m \text{ is chosen,} \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{i,j}^{e,m} = \begin{cases} 1, & \text{if } e \text{ is mapped on substrate link } (i,j) \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{i,j}^{\hat{e},m} = \begin{cases} 1, & \text{if } \hat{e} \text{ is mapped on substrate link } (i,j) \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$$

$t_{i,j}$: the primary traffic reserved on substrate link (i, j).
$\hat{t}_{i,j}$: the backup traffic reserved on substrate link (i, j).

- Mathematical Model:

$$Min \sum_{(i,j)\in L} (t_{i,j} + \hat{t}_{i,j})$$

Subject to

$$\sum_{m\in\mathcal{M}} x_m = 1 \tag{4.2}$$

$$\sum_{j:(i,j)\in L} y_{i,j}^{e,m} - \sum_{j:(j,i)\in L} y_{j,i}^{e,m} \begin{cases} \leq 1 & \text{if } \delta_{s(e),i}^{m} = 1, \\ \geq -1 & \text{if } \delta_{d(e),i}^{m} = 1, \\ = 0 & \text{otherwise} \end{cases} \quad \forall\ i \in N,\ e \in E,\ m \in \mathcal{M}. \tag{4.3}$$

$$y_{i,j}^{e,m} \leq x_m \quad \forall e \in E, m \in \mathcal{M}, (i,j) \in L. \tag{4.4}$$

$$t_{i,j} = \sum_{m\in\mathcal{M}} \sum_{e\in E} y_{i,j}^{e,m} d_e \quad \forall (i,j) \in L. \tag{4.5}$$

$$\sum_{j:(i,j)\in L} y_{i,j}^{\hat{e},m} - \sum_{j:(j,i)\in L} y_{j,i}^{\hat{e},m} \begin{cases} \leq 1 & \text{if } \delta_{s(\hat{e}),i}^{m} = 1, \\ \geq -1 & \text{if } \delta_{d(\hat{e}),i}^{m} = 1, \\ = 0 & \text{otherwise} \end{cases} \quad \forall\ i \in N,\ \hat{e} \in \hat{E},\ m \in \mathcal{M}. \tag{4.6}$$

$$y_{i,j}^{\hat{e},m} \leq x_m \quad \forall \hat{e} \in \hat{E}, m \in \mathcal{M}, (i,j) \in L. \tag{4.7}$$

$$\sum_{m\in\mathcal{M}} \sum_{\hat{e}:\in BG(v)} y_{i,j}^{\hat{e},m} d_{e:\{v,d(\hat{e})\}} - \sum_{m\in\mathcal{M}} \sum_{e:\in WG(v)} y_{i,j}^{e,m} d_e \leq \hat{t}_{i,j} \quad \forall (i,j) \in E, v \in V \tag{4.8}$$

$$t_{i,j} + \hat{t}_{i,j} \leq d_l \quad \forall l : (i,j) \in L. \tag{4.9}$$

$$x_m,\ y_{i,j}^{e,m},\ y_{i,j}^{\hat{e},m} \in [0,1] \quad \forall m \in \mathcal{M}, e \in E, \hat{e} \in \hat{E}, (i,j) \in L. \tag{4.10}$$

$$t_{i,j},\ \hat{t}_{i,j} \geq 0 \quad \forall (i,j) \in L. \tag{4.11}$$

We aim at minimizing the overall bandwidth cost for the given SVN mapping solution. This encourages the model to select a node mapping solution where the nodes are not too widely spread. Hence, we set the model's objective function to minimize the sum of primary and backup traffic on the substrate links. Constraint (4.2) forces the model to select a single node mapping solution. Constraint (4.3) represent the flow conservation constraint for the primary virtual links. Constraint (4.4) indicates that a primary link mapping solution will only be constructed for the chosen node mapping solution. Constraint (4.5) measures the primary traffic routed on every physical link in the substrate network. Constraint (4.6) represents

the flow conservation constraints for the backup virtual links. Constraint (4.7) indicates that a backup link mapping solution will only be constructed for the chosen node mapping solution. Constraint (4.8) measures the backup link traffic routed on every physical link in the substrate network by exploiting cross-sharing and backup-sharing. Constraint (4.9) ensures that the sum of the primary and backup bandwidth routed on each substrate link does not violate its capacity.

4.7 The SVNE Heuristic (SVNE-H)

Given the NP-Hard nature of the SVNE problem, we have devised an embedding heuristic that relies on the weighted-Dijkstra algorithm for the routing of the primary and backup links. Similar to the SVNE model presented in Section 4.6.1, the SVNE heuristic also employs a two-step VNE approach, that begins by generating a list of $\mathcal{M}$ node mapping solutions using the Virtual Machine Placement (VMP) algorithm [49]. Next, for each node mapping solution $m \in \mathcal{M}$, the SVNE heuristic is executed; the algorithm terminates at the occurrence of the first node mapping solution that yields a feasible link embedding solution.

The procedural details of the SVNE heuristic are presented in Algorithm 4.3. It consists of two main steps; in Step 1, the primary virtual links are embedded using the *getShortestPath*, and in Step 2 the backup virtual links are embedded. The benefit of embedding the primary links first is to encourage the backup-link to be routed through the same physical links, and re-use the primary bandwidth via cross-sharing. Therefore, for each backup link, the algorithm detects the set of virtual nodes the former protects (Lines 20-25). This implies that this backup link can cross-share with the working path of each of these virtual nodes (denoted by T). Subsequently, to encourage cross-sharing, the weight on each physical link $l \in T$ is set to 0 (Line 26-32), and the *getShortestPath* algorithm is executed to run the weighted-Dijkstra algorithm. The $getShortestPath$ method presented in Algorithm 4.4 consists of running a simple weighted-Dijkstra algorithm for primary links (Lines 24-30). When a path p is found, the physical links that compose p are checked to ensure that they satisfy the bandwidth requirement of the virtual link e; if a bandwidth violation occurs, the infeasible path found is added to a dedicated set T that maintains the list of infeasible paths found, and the weight on the physical links composing this invalid path is incremented by a large number (equal to the number of physical links in the substrate network) (Lines 32-34). The algorithm terminates when a feasible path is found, or in the event where an infeasible path is generated twice.

Algorithm 4.3 SVNE-Heuristic(NodeMappingSolution m)

Given
$G^v = (V, E)$ /*Virtual Network*/
$G^s = (N, L)$ /*Substrate Network*/
$\hat{E}$ /*Set of Virtual Links*/
$P = \{\}$; /*Set of Primary Paths*/
$\hat{P} = \{\}$; /*Set of Backup Paths*/
Step 1: Embed Primary Links $\mathcal{M}$
for $(e \in E)$ **do**
    /*Initialize Link Weights*/
    for $(l \in L)$ **do**
        l.weight = 1;
    end for
    $p = getShortestPath(e,P,\hat{P},V)$;
    if $(p ==$ NULL$)$ **then**
        Return NULL;
    end if
    $P = P \cup p$;
end for
Step 2: Embed Backup Links $\mathcal{M}$
for $(\hat{e} \in \hat{E})$ **do**
    $T = \{\}$;
    for $(v \in V)$ **do**
        if $(\hat{e} \subset BG_v)$ **then**
            $T = T \cup getWorkingPaths(v, P)$;
        end if
    end for
    for $(l \in L)$ **do**
        if $(l \in T)$ **then**
            l.weight = 0;
        else
            l.weight = 1;
        end if
    end for
    $\hat{p} = getShortestPath(\hat{e},P,\hat{P},V)$;
    if $(\hat{p} ==$ NULL$)$ **then**
        Return NULL;
    end if
    $\hat{P} = \hat{P} \cup \hat{p}$;
end for
Return $MappingSolution(m, P, \hat{P})$;

Algorithm 4.4 getShortestPath(virtual_link e, path_set P, path_set $\hat{P}$, virtual_nodes V)

```
1: T ={}; /*Tentative Paths*/
2: p = weighted-Dijkstra(e);
3: while !(T.contains(p)) do
4:     if (isBackupLink(e)) then
5:         for (l ∈ p) do
6:             bw = 0
7:             for (v ∈ V) do
8:                 if ( e ⊂ BG_v) then
9:                     resv = getVLink(e.destination,v).bw;
10:                    /*Apply Cross-Share*/
11:                    resv -= doCrossShare(v,P);
12:                    /*Apply Backup-Share*/
13:                    resv -= doBackupShare(!v,P̂);
14:                    bw = Max(bw,resv );
15:                end if
16:            end for
17:            if (l.bw ≤ bw) then
18:                T = T ∪ p;
19:                Break to line 32;
20:            end if
21:        end for
22:        Return p;
23:    else
24:        for (l ∈ p) do
25:            if (l.bw ≤ e.bw) then
26:                T = T ∪ p;
27:                Break to line 32;
28:            end if
29:        end for
30:        Return p;
31:    end if
32:    for (l ∈ p) do
33:        l.weight = l.weight + |L|;
34:    end for
35:    p = weighted-Dijkstra(e);
36: end while
37: Return NULL;
```

However, when embedding a backup link, cross-sharing and backup-sharing must be performed to minimize the amount of bandwidth that needs to be provisioned. Here, every virtual node v that contains e in its backup group demands a different amount of backup bandwidth, denoted as $resv$; Subsequently, if any of the working paths of v coincide with any physical link in p, cross-sharing can be performed (Line 11). However, given the fact that the link embedding is performed sequentially, attention must be placed to ensure that no two backup links of v are cross-sharing the same bandwidth. Similarly, if any of the backup paths that do not belong to the BG_v share common substrate links, then backup-sharing can be performed, while avoiding redundant backup-sharing with other backup links in BG_v (Line 13). Let k be the maximum number of iterations that the $getShortestPath$ algorithm executes before running into a redundant path, then the complexity of the SVNE-Heuristic is $\mathrm{O}(k \cdot |E| \cdot |\hat{E}| \cdot |N|^2)$, where $\mathrm{O}(|N|^2)$ is the runtime complexity of the weighted-Dijkstra Algorithm.

4.8 Numerical Results

4.8.1 Performance Evaluation

Table 4.1: Execution Time (sec)& Cost for 1 VN - FatTree ($K = 8$)

T	ProRed-M		1Red-M		KRed-M		ProRed-H		1Red-H		KRed-H	
	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time
5	1090	1.6	1265	1.4	1720	1.3	1090	0.6	1265	0.2	1720	0.45
10	3167.5	2.3	3370	1.9	5530	2.7	3192.5	1.1	3390	0.8	5530	3.71
15	3982.5	2.7	4022.5	2.1	6620	5.0	4012.5	1.8	4022.5	7.4	6737.5	2.58
20	8965	7.4	10880	82.8	16275	541.6	9185	6.0	11157.5	5.1	16497.5	14.80
25	11417.5	68.5	13100	6332.3	19992.5	13500	11682.5	9.7	13485	7.5	20458.5	21.99

We start by evaluating the performance of the SVNE-Heuristic against the SVNE-Model over the FatTree ($k = 8$) network [14]. To do so, we look at the execution time and the cost achieved by the different redesign techniques as we vary the size of the VN. The aim of this test is to see how well each of these embedding techniques scale, and how reliable our SVNE-Heuristic is in terms of the total achievable bandwidth cost. Herein after, we refer to the combination of a particular redesign technique followed by a particular embedding method by *RedesignTechnique-M* when the redesigned SVN is mapped using the SVNE-Model, and *RedesignTechnique-H* when the redesign is followed by the SVNE-Heuristic; e.g. $ProRed-M$ refers to a VN that was redesigned into a SVN using ProRed, followed by the

SVNE-Model for the embedding step.

Looking at Table 4.1, we observe that the execution time of the SVNE-Model undergoes a steep increase as the size of the VN grows, where embedding a 1-redundant SVN of 25 receivers takes up to 6332.3 seconds, whereas embedding a k-redundant SVN of 25 receivers requires 13500 seconds, which is highly impractical. Further, we notice that embedding a ProRed SVN requires almost a minute, which is far more scalable than that of 1- and k-redundant redesigns, yet it remains nonviable for an online embedding scheme. The reason why ProRed's redesign yield a faster embedding is because the size of the problem does not grow as fast as it does when dealing with the existing redesigns, since ProRed limits the number of backup links by adopting the sets creation practice.

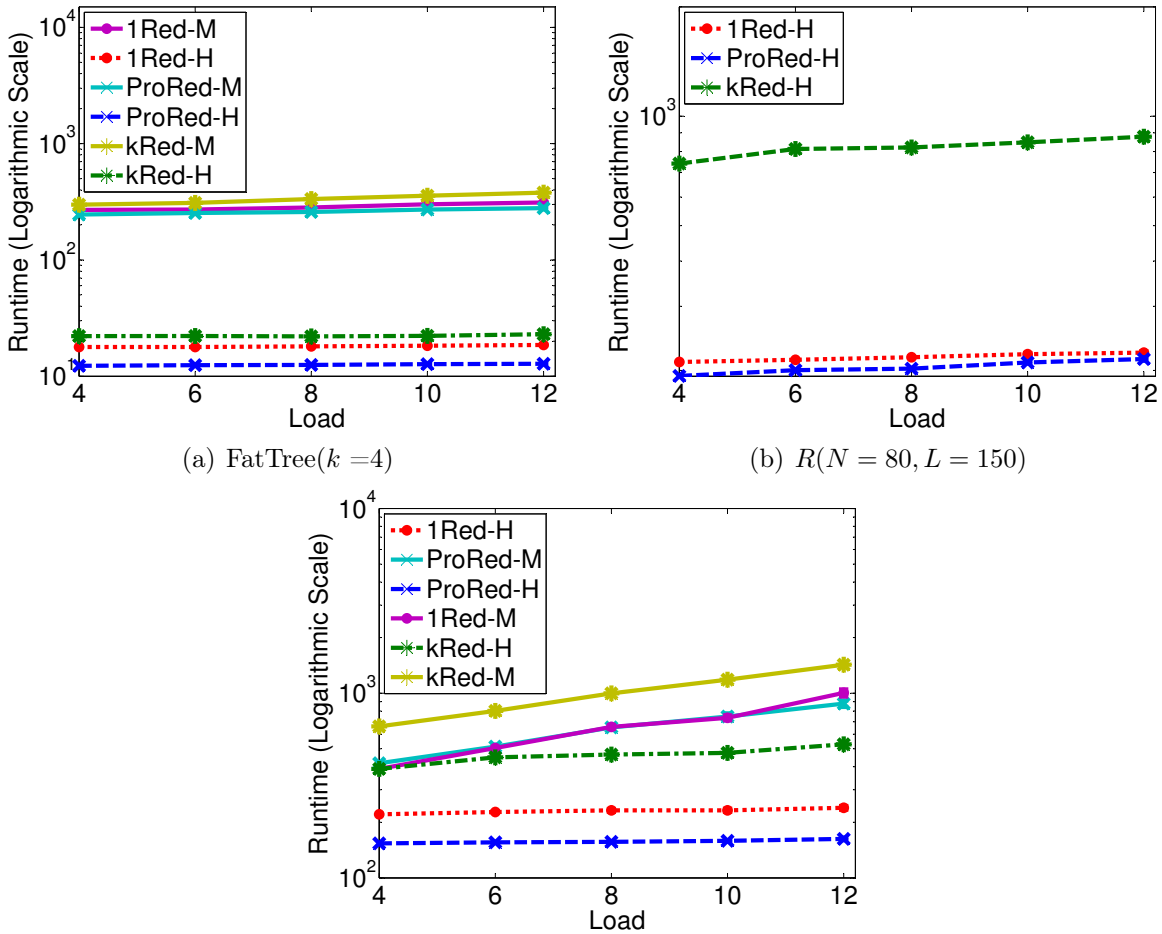
Second, we observe that the cost achieved by the SVNE-Heuristic is highly comparable to that achieved by the SVNE-Model, with a gap that does not exceed 3% for all of the redesign techniques. Finally, we observe that embedding cost achieved by the ProRed redesign is lower than that achieved by either 1- or k-redundant. In fact, ProRed-M achieves 12% lower cost than 1Red-M and 42% lower cost than KRed-M. Similarly, ProRed-H achieves 13% lower cost than 1Red-H and 42% lower cost than KRed-H.

4.8.2 Comparative Analysis

Now we perform a comparative analysis of ProRed against the 1-redundant and k-redundant schemes for various metrics: Blocking Ratio, Cost-to-Revenue Ratio, Total Revenue, and Execution Time. Here, blocking ratio refers to the percentage of VNs rejected out of the total number of VN embedding requests received at the end of each test case. We adopt 4 different substrate network topologies to conduct this evaluation. The substrate networks used for our simulation are FatTree (k=4) and FatTree (k=8) [14], in addition to the JellyFish (k =10) [117] network topology, and a randomly generated network [118] R with 80 nodes and 150 links. In all of these substrate networks, we set the CPU capacity of each host node to 64 units, and the bandwidth capacity on the substrate links is set to 1000 units. We perform the redesign and mapping of VNs in an online fashion, upon the arrival of each request. As VNs arrive and leave, the load on the network varies, thereby changing the state of the substrate network in terms of residual capacity. Our proposed redesign and embedding schemes thereby operate under this load-varying environment, wherein the state of the network varies, and the set of incoming VN requests overtime is unknown. The size of each VN can range between [4-12] virtual nodes when using FatTree (k=4), and between [10-30] virtual nodes when running atop FatTree (k=8), JellyFish (k=10) or R. Each virtual

node can be connected to any other virtual node in the VN request with a probability of 50%. The CPU demand of the virtual nodes is set to be in the range [4-12], and the bandwidth demand on the virtual links is in the range [50-100]. Further, we set the size of node mapping solutions $\mathcal{M}$ to 30. In all test cases, the results are averaged over 15 runs, and we show the margin of error with a 95% confidence.

1. **Average Execution Time :** First, we look at the average runtime to redesign and embed



(a) FatTree(k =4)

(b) $R(N = 80, L = 150)$

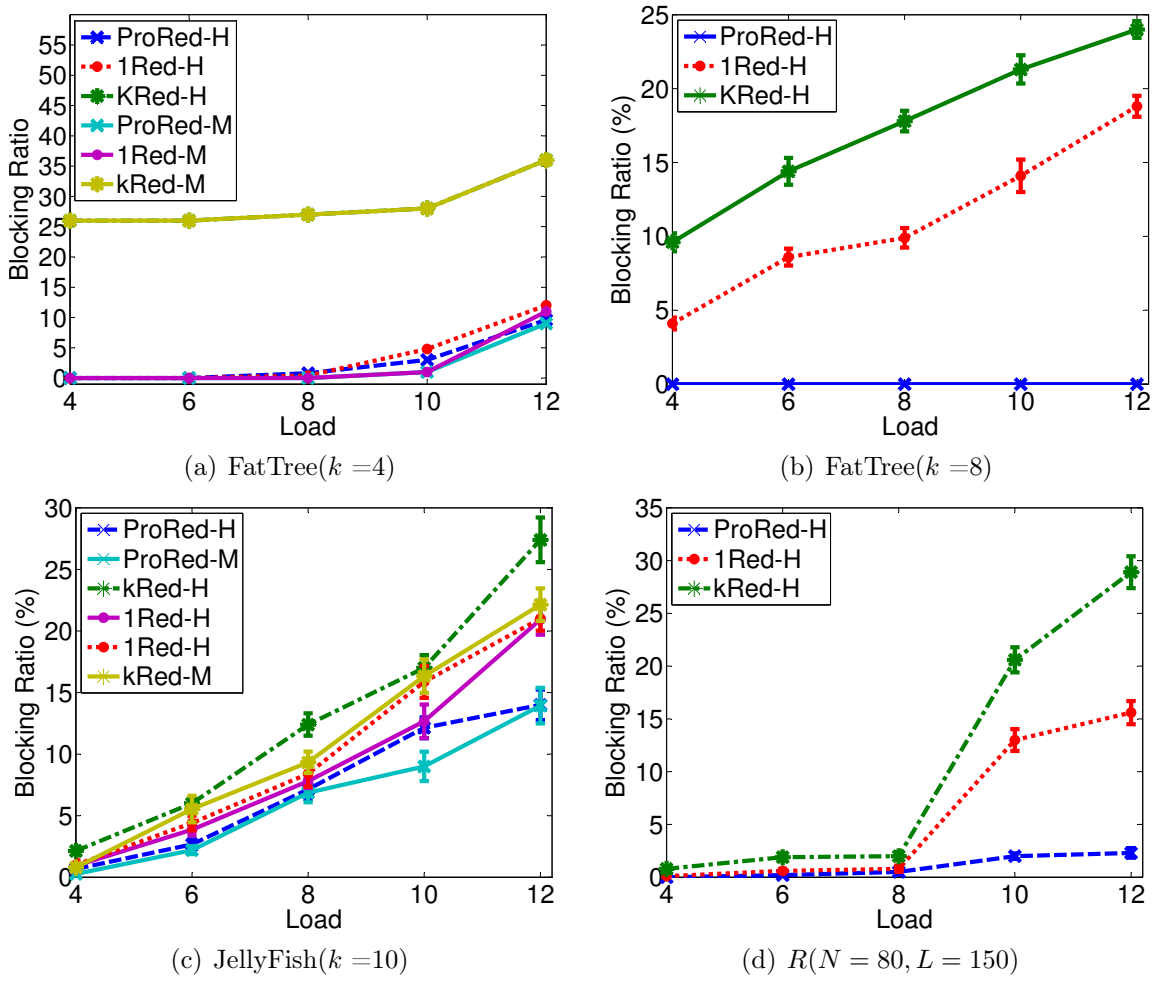
(c) JellyFish(k =10)

Figure 4.8: Execution Time

a single SVN using the various aforementioned redesign and embedding techniques; the results are illustrated in Figure 8(a), 8(b), and 8(c), respectively. Here again, we assert the scalability of the SVNE-Heuristic against the SVNE-Model, and we observe that as the load increases, ProRed-H outperforms KRed-H and 1Red-H in terms of average execution time.

2. **Blocking Ratio :**

Now, we evaluate the blocking ratio over the FatTree(k=4) and FatTree(k=8), as well as



(a) FatTree(k =4)

(b) FatTree(k =8)

(c) JellyFish(k =10)

(d) $R(N = 80, L = 150)$

Figure 4.9: Blocking Ratio

JellyFish (k=10) and the random topology R. The results are shown in Figure 4.9. We observe for all substrate network topologies, ProRed yields the lowest blocking ratio. For instance in FatTree(k=4), ProRed-M achieves 25% lower blocking that 1Red-M and 78% lower blocking that KRed-M for a load of 12 (Figure9(a)). Similarly, ProRed-H achieves 28% lower blocking than 1Red-M, and 77% lower blocking than KRed-M for a load of 12. This gain is mainly attributed to ProRed's ability to explore the space between 1 and k. Since FatTree connects each host node to the substrate network with a single substrate link, this architecture puts 1-redundant at a great disadvantage, as the backup node is forced to go through a single substrate link in order to reach the neighbors of all the critical nodes in a given VN. Though k-redundant does not concentrate the backup bandwidth

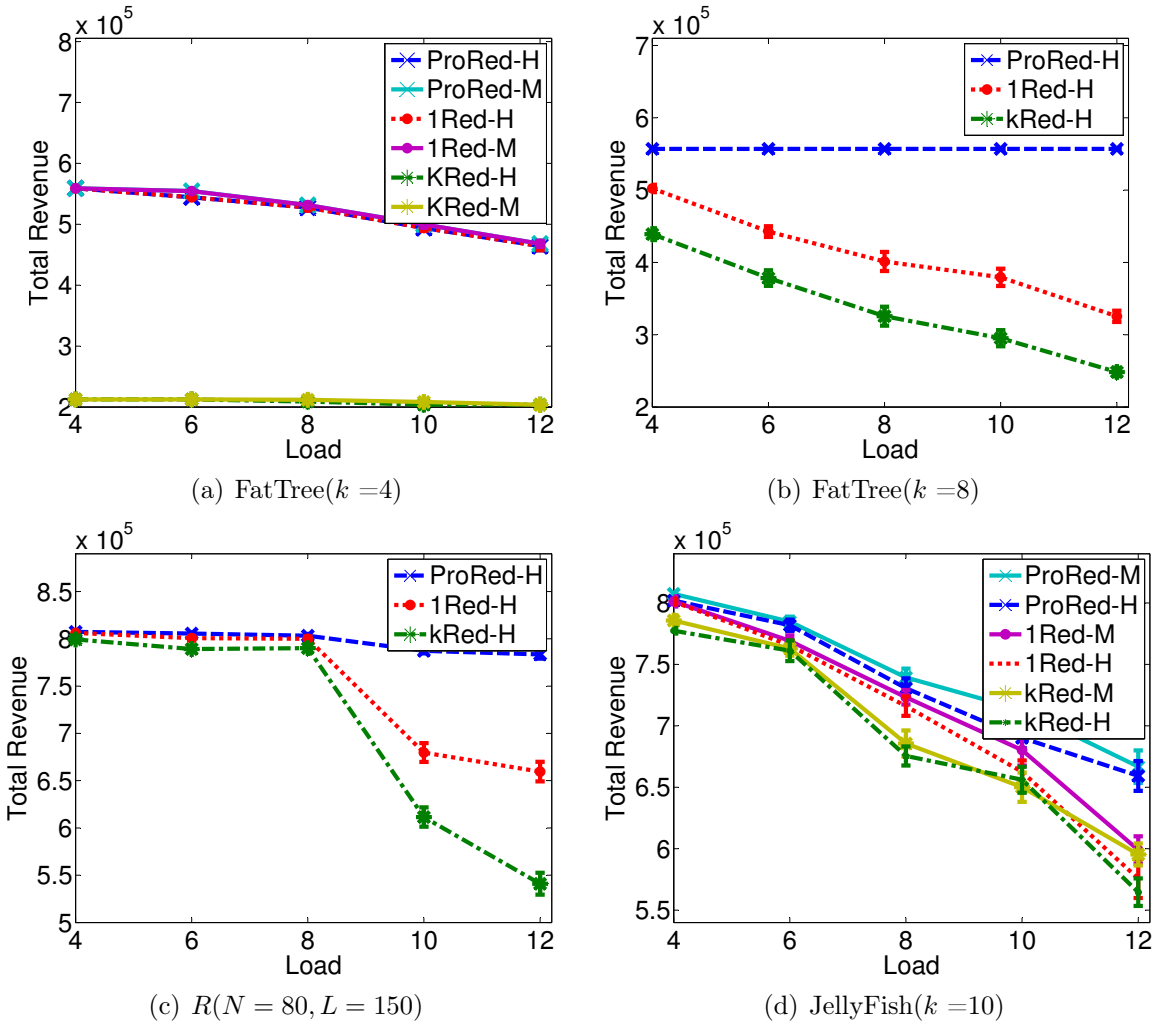
load on a single substrate link, its redesign technique requires as many backup nodes as the number of critical nodes in a VN request, which renders a substantial amount of CPU and bandwidth demand to associate each backup node with its corresponding primary virtual node. Whereas ProRed maintains a balance between the number of allocated backup nodes and links, thus its blocking ratio prevails over its peers. Similar gain is observed on FatTree(k=8)(Figure9(b)).

Given that the FatTree topology does not allow ProRed to employ its prognostic redesign technique, we further compare these 3 redesign techniques over JellyFish ($k = 10$) and R to evaluate the advantage of this property. We observe that ProRed achieves encouraging gain in terms of decreasing the blocking ratio. We find that as we increase the load to 12, ProRed-H achieves 44% lower blocking than 1Red-H and 55% lower blocking than KRed-H (Figure 9(c)). Indeed, the rich interconnection of the random network topology enables ProRed from exercising its prognostic redesign technique. Hence, ProRed is capable of greatly decreasing the incurred bandwidth cost for each VN, and subsequently increasing the network's admissibility. Similar gain is observed on R as shown in Figure 9(d).

3. **Revenue :**

   Revenue is an important metric that highly complements the blocking ratio metric. A low blocking ratio does not necessarily indicate a high revenue. This is because the concerned model may only be capable of admitting VNs with lower resource demands. When in fact, VNs with substantial resource requirements are more profitable to the cloud provider. In this regard, we measure the total revenue obtained by the various redesign techniques. Given that the aim of the metric is to evaluate each of the aforementioned techniques's ability to admit profitable VNs, we measure the revenue of each VN in function of its overall CPU demands and size using the following equation: Revenue $= \sum_{v \in V} c_v + \pi_v$ |V|. The results are shown in Figures 10(a)-10(d). Once more we observe for FatTree(k=8), ProRed achieves encouraging results, with a 51% gain over 1-redundant and 62% gain over k-redundant for a load of 10 (Figure 10(b)). Similar results are observed over all other network topologies. This gain is mainly attributed to ProRed's unique redesign properties, which significantly reduce the average cost, and hence leverage the efficient utilization of the substrate network. Subsequently, ProRed is capable of admitting more profitable VNs in comparison with the 1 and k-redundant schemes.

4. **Cost-to-Revenue Ratio :** For a given VN, the cost is measured using the objective function of the SVN embedding model presented in Section V, which is the sum of the



(a) FatTree(k =4)

(b) FatTree(k =8)

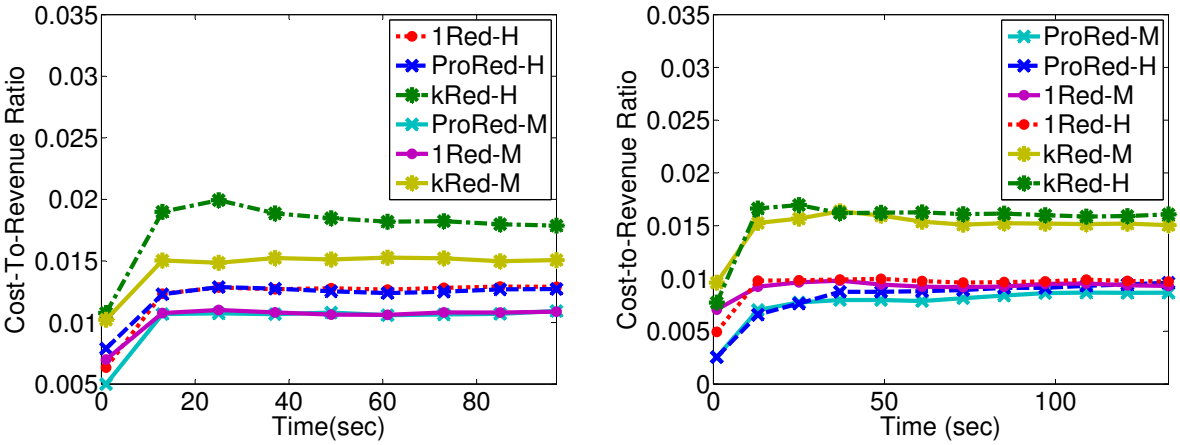
(c) $R(N = 80, L = 150)$

(d) JellyFish(k =10)

Figure 4.10: Total Revenue

primary and backup bandwidth incurred by this VN in the substrate network. For each of the aforementioned redesign techniques, we measure the cost-to-revenue ratio over time, and compare the results obtained by the SVNE-Model against the SVNE-Heuristic. The cost-to-revenue ratio indicates the cost incurred to generate 1 unit of revenue. We perform this evaluation over FatTree (k=4) and JellyFish (k=10) network topologies, the results are illustrated in Figures 11(a) and 11(b), respectively. Clearly we observe that the cost-to-revenue ratio achieved by ProRed-M and ProRed-H outperforms the rest of the redesign techniques. ProRed's prognostic redesign technique for backup resource sharing enables it to achieve this gain,while 1-redundant and k-redundant falls short due to their agnostic approach.

5. **Impact of Network Density:**



(a) FatTree(k =4) (b) JellyFish(k =10)

Figure 4.11: Cost-to-Revenue Ratio Over Time

Finally, we look at the impact of the network density on the various redesign techniques.

Table 4.2: Blocking for 100 VNs - R(N= 50,L)

# L	ProRed		1Red		KRed	
	Blocking Ratio	Total Revenue	Blocking Ratio	Total Revenue	Blocking Ratio	Total Revenue
80	27	243000	34	173800	37	139000
100	15	367800	30	213800	33	181000
200	3	493800	13	385600	17	349800
400	0	525200	0	525200	0	525200

To do so, we randomly generate [118] a network of 50 nodes and vary the number of substrate links from [80-400] while fixing the load to 12. The results are presented in Table 4.2. Naturally, the blocking incurred by each of the aforementioned redesign techniques decreases as the network density increases. Yet we observe that the blocking and revenue incurred by ProRed outperforms its peers at every instance. For instance when the network contains 200 links, ProRed achieves 76% and 82% lower blocking that 1-redundant and k-redundant (respectively), with at least 22% higher revenue gain than both.

4.9 Conclusion

In this chapter, we presented ProRed a novel prognostic redesign technique for survivable virtual networks against single facility node failures. ProRed goes beyond the dogmatic redesign techniques that fix the number of backup nodes to either 1 or k. Further it is equipped with a unique property that enables it to design SVNs that can highly promote backup resource sharing once embedded in the substrate network. This property lies in

positioning of the backup nodes in the SVN such that backup-sharing and cross-sharing can be fully exploited. We compared ProRed against 1-redundant and k-redundant schemes, and we show that it achieves significant gains in terms of decreasing the blocking ratio, achieving lower average cost and substantially higher revenue, in considerably lower execution times.

Chapter 5

Post-Failure Restoration for Multicast Services in Data Center Networks

5.1 Problem Statement

As discussed in Chapter 3, many applications and services [72–74, 77, 88–90] hosted in cloud data center networks today rely on multicast communication to disseminate traffic. The failure-prone nature of data center networks has evoked countless contributions [116] from the research community to develop proactive and reactive countermeasures. Yet, most of these aforementioned techniques were developed with unicast services in mind, thereby fail to cater to the distinctive properties and QoS requirements that multicast services entail. Multicast services differ from unicast VNs in many aspects, which ultimately inhibit the applicability of the existing protection schemes. As presented in Chapter 3, a Multicast Virtual Network (MVN) comprises two types of Virtual Machines (VMs): the multicast source and a set of multicast recipient nodes. Further, the routing of traffic consists of building a multicast distribution tree between the multicast source and receivers (recipient nodes) in order to avoid duplicate traffic. Also, multicast services which involve real-time communication entail stringent QoS requirements, such as end-to-end delay and delay-variation (differential-delay) constraints. Thus, the problem of survivable MVNs in the event of failures demands a separate attention and a tailored restoration scheme that responds to its distinctive properties. In this regard, this chapter is devoted towards understanding the impact of failure on multicast services residing in data center networks. We build on our previous work presented in Chapter 3 which developed an efficient technique for solving the MVN placement problem, and we assume that the MVNs are already residing in the data center network. We

begin by studying the impact of failures on MVNs under various failure scenarios; then, we focus the scope of this work to study the problem of survivable MVNs in the case of facility node failures. Here, we present a formal definition of the MVN restoration problem in the event of facility node failure, and we prove its NP-Complete nature in general graphs (e.g. inter-data center network interconnects, or wide-area networks). Further, we exploit the structure topology of data center network to prove that the problem at hand can be solved in polynomial-time for multi-rooted tree-like data center network topologies.

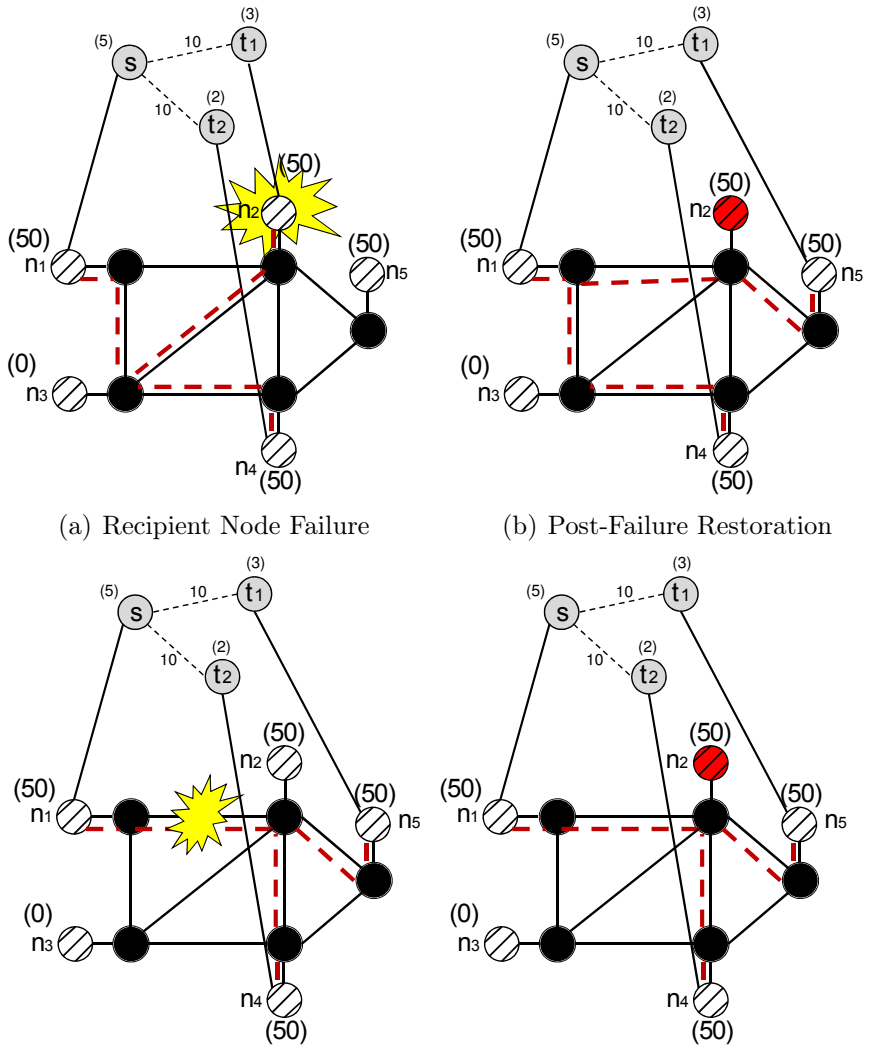
5.2 Related Work

Given the failure-prone nature of data center networks, a handful [73, 74, 77] of contributions in the literature has surfaced to provide reliable multicast services. For instance, the authors in [73] considered the problem of physical link or network node failures, and proposed the use of backup overlays to retransmit traffic to affected nodes in a peer-to-peer fashion. In [77], the authors considered the problem of provisioning survivable multicast VNs with end-to-end delay constraints against single facility node failures. However, the authors assumed that the failure would only affect recipient nodes, and thus the problem of restoring a MVN source node has been disregarded. The same authors [74] have also tackled the problem of survivable MVNs in the event of a single regional failure. Here, the authors proposed to handle the problem in a proactive fashion by augmenting the MVN with backup-nodes. However, one common limitation with proactive protection schemes is the fact that the provisioned resource will remain idle until a failure occurs, which yields poor network resource utilization.
Our work is different since we consider the problem of restoring delay-sensitive MVNs against facility node failures, and we tackle the problem in a reactive fashion such that any failure can be restored while maintaining the QoS of the working multicast tree. Further, we prove that the problem can be solved to optimality in polynomial-time when applied to multi-rooted tree-like data center network topologies. Accordingly, these proposed contributions distinguish our work from existing literature.

5.3 Impact of Failure on Multicast Services

5.3.1 Impact of Failure on MVNs

As we have previously mentioned, failures in data center networks can either affect a substrate facility node (e.g., physical server), a substrate link, or a network node (e.g., a router or



(a) Recipient Node Failure

(b) Post-Failure Restoration

(c) Tree Link Failure

(d) Tree Maintenance

Figure 5.1: Impact of a Substrate Node or Physical Link Failure

a switch). One way to protect a multicast VN against a facility node failure is by augmenting the latter with backup nodes, and then embed the resultant graph onto the substrate network while provisioning enough backup resources. As for the failure of substrate links, this can be mitigated by constructing an edge-disjoint backup tree. Such a scheme is commonly known as proactive protection, since the backup nodes and links are instated prior to any failure [79, 119]. While this offers a certain degree of reliability, it is also fairly costly since the provisioned resources for these backup nodes and links remain idle until failures occur. An alternative approach could be to restore the affected resource(s) upon failures. Such a "reactive approach" is more cost-efficient as it eliminates idle resources in the network, but it demands fast restoration time to avoid long service downtime. This work focuses on

providing reactive countermeasures.

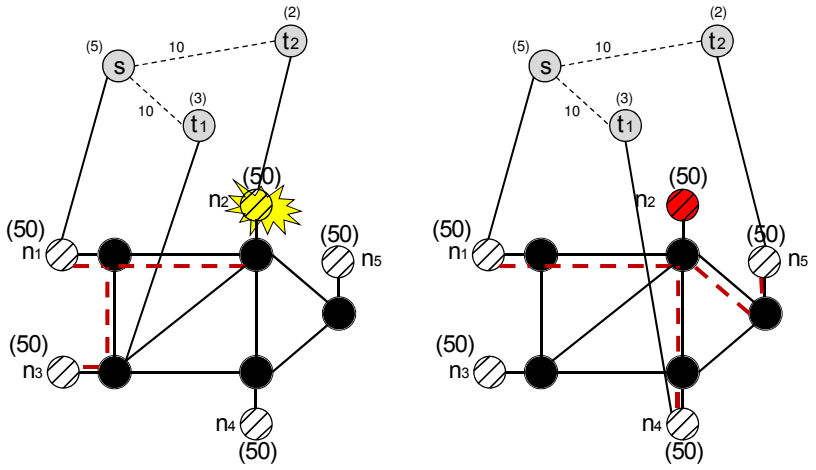
Case of Facility Node Failure: When a failure affects a facility node hosting a recipient VM, the restoration scheme necessitates finding a backup that can host the failed VM with sufficient resources. In addition, when the failed receiver belongs to a delay-sensitive MVN, the path used to connect the backup to the rest of the multicast tree must also maintain the MVN's QoS requirements; that is it must be within the end-to-end delay constraint, and satisfies the differential-delay with the remaining working receivers. Figure 1(a) illustrates the case of a 2-receivers MVN hosted in a data center network; where the failure of facility node n_2 brought down one of its recipient nodes (t_1). Given the substrate network's capacity, n_5 is the only substrate node that has enough resources to host t_1. Now to connect t_1's new host to the rest of the MVN, the path used to reach n_5 must be within the end-to-end delay constraint of 2, and satisfies the differential delay to the rest of the working receivers; the working receiver in this case is t_2. Hence, we need to connect n_5 to the remaining working tree with exactly 2 hops given the differential-delay constraint of 0 for the MVN in question. Subsequently, the only feasible restoration solution in this case is to connect n_5 to the multicast tree via substrate path $\{n_1$-n_2-$n_5\}$, as illustrated in Figure 1(b). On the other hand, when a failure affects a substrate node hosting the source of a MVN, it mandates a look-up for a backup node that can host the failed source, as well as a multicast tree reconstruction that spans all existing receivers and respects the QoS requirements.

Case of Substrate Link or Network Node Failure: In the event of a substrate link failure or a network node failure, this latter will detach an entire subtree, thereby disconnecting one or many nodes connected via this subtree to the rest of the multicast service. Figure 1(c) illustrates the case where substrate link $\{n_1 - n_2\}$ fails, thereby detaching the subtree rooted at n_2's adjacent network node, disconnecting receivers t_1 and t_2. Similar outcome will occur if the network node connected to n_2 fails.

When restoring a MVN, it is insufficient to find the backup node that maintains the service's QoS; rather it is important to also consider the cost of the resultant tree. It is in the network provider's best interest to minimize the embedding cost of the hosted services in the aim of maximizing both his/her revenue, as well as the network's admissibility. For instance, after restoring receiver t_1 post-failure of its original host n_2, the resultant tree shown in Figure 1(b) is more costly than the pre-failure multicast tree of the given MVN. An alternative solution (illustrated in Figure 1(d)) could be to re-route the traffic to t_2 via substrate path $\{n_1 - n_2 - n_4\}$, thereby maintaining the MVN's QoS requirements while achieving a lowest-cost tree. In light of the above, we can conclude the following key observation:

Observation: Multicast VN restoration demands both a service repair to restore the failed element, as well as a multicast tree maintenance to reconstruct the lowest-cost tree that maintains the requested QoS.

5.3.2 Advantage of Migration-Aware Restoration Schemes



(a) No Feasible Restoration (b) Migration-Aware Restoration

Figure 5.2: Advantage of Recipient Nodes Migration

Now, it could happen that the network operator fails to find a way to restore a service while maintaining its QoS requirements. To illustrate this, recall the 2-receivers MVN hosted in a 5-nodes substrate network as shown in Figure 2(a). The failure of recipient node t_2 kicks off a reactive look-up for a valid backup node that can assume the role of t_2 and restore the affected service. Given that the MVN in question has an end-to-end delay requirement of 2, then t_2 can migrate to substrate nodes n_4 or n_5 while satisfying the end-to-end delay constraint. However, with the additional differential-delay constraint of 0, there are no possible ways to restore t_2 while maintaining the requested QoS.

One possible solution for this problem could be to re-embed the failed MVN from scratch. Such an approach is seemingly unpleasant as it disrupts the entire service. A more promising solution is to encourage *migrating* some parts of the working MVN to widen the search space. Figure 2(b) highlights this advantage. Clearly, by migrating the active receiver VM t_1 to substrate node n_4, it is now possible to migrate the failed receiver VM t_2 onto substrate node n_5 and reconstruct a multicast tree that interconnects both receivers to the source while satisfying both end-to-end delay and differential-delay requirements.

5.4 The MVN Restoration Problem against Facility Node Failure

In this work, we consider the case of a facility node failure. Below we provide a formal definition of the MVN restoration problem in the event of a facility node failure by presenting first an overview of the network model.

5.4.1 Network Model Overview

In this chapter, we adopt the same network model as the one presented in Chapter 3 under Section 3.3.1; that is we repersent the substrate network as an undirected capacitated graph denoted by $G^s = (N,L)$, and a delay-constrained MVN request denoted by $G^v = (s, T, b', \gamma, \delta)$. Recall that s represents the multicast source, T the set of recipient nodes, b' the bandwidth demand, and γ and δ represent the end-to-end delay and differential-delay, respectively.
Further in this chapter, we assume that the MVN requests are already residing in the substrate network using one of the MVNE embedding schemes presented in Chapter 3.

5.4.2 Multicast Virtual Network REstoration Model (REM)

In this section, we present REM: a MVN restoration model. Given the advantage of migration, we equip REM with the ability to migrate some (or all) parts of the MVN service in the attempt to reconstruct a lowest-cost tree. REM achieves the following two objectives:

- **MVN Repair:** consists of restoring the failed service component, be it failure of the source node, any recipient node, or a physical link in the multicast tree. Repairing physical link failures is performed by translating a link failure into the appropriate set of disconnected receivers.

- **Tree Maintenance:** ensures that the repair does not violate the requested QoS, and yields a lowest-cost tree.

REM assumes as input a substrate network, and the MVNE solution ($x^0_{v,n}$ indicates the node mapping solution) for the failed multicast service(s). Table 5.1 presents a description of the inputs and decision variables used in our problem formulation. When a failure occurs, the restoration model is invoked for each affected MVN in the aim to repair the affected service components, and restore a low-cost delay-bounded multicast tree. Given that VM migration

Table 5.1: Notations Description

Inputs	
$\widetilde{N}$	Set of failed facility nodes.
N'	$= N$ - $\widetilde{N}$, Set of active facility nodes.
α	$\in$ [0,1] allows to adjust the weight between the dual objectives.
$x^0_{v,n}$	$\in$ [0,1], indicates whether VM v is placed on substrate node n (pre-restoration node mapping solution).
c'_v	Demand of VM v.
c_n	Capacity of substrate node $n \in N$
V	Set of all VMs {s,T} in a MVN.
b'	Demand of the virtual links in MVN request.
b_l	Capacity of substrate link $l \in L$.
γ	End-to-End delay constraint of the MVN.
δ	Differential delay constraint of the MVN.
Decision Variables	
$t'_{i,j}$	≥ 0, denotes the traffic provisioned on substrate link (i, j).
ς_v	≥ 0, denotes the cost of migrating VM v.
ρ_v	$\in$ [0,1], denotes whether VM v migrated or not.
$y^v_{n,n'}$	$\in$ [0,1], indicates whether VM v migrates from n to n'
$x^1_{v,n'}$	$\in$ [0,1], indicates whether VM v is placed on substrate node n' in the post-restoration node mapping solution.
r_n	≥ 0, denotes the post-migration residual capacity on substrate node $n \in N$
$q^v_{i,j}$	$\in$ [0,1] indicates whether the path from the source to VM v is routed through link (i, j)
θ_{min}	≥ 0, measures the minimum differential-delay.
θ_{max}	≥ 0, measures the maximum differential-delay
$z_{i,j}$	$\in$ [0,1], indicates whether link (i, j) is part of the MVN tree.

has an associated cost and bandwidth usage, our proposed technique aims at restoring the failed multicast service with the least migration cost, while attempting to reconstruct a lowest-cost tree. Our objective function is presented in Equation 5.1. Typically, migration cost is a function of the load on substrate links used to perform the migration [120], since the latter reflects the downtime experienced by the MVN [121]. Given that computing migration cost in terms of load on paths used for migration will render a non-linear objective function, we linearize our objective function by assuming that the load in the network is uniform, and we consider the number of migrations performed as a reflection of the migration cost. The remainder of our mathematical formulation is presented below:

$$Minimize\ \alpha(\sum_{(i,j)\in L} t'_{i,j}) + (1-\alpha)(\sum_{v\in\{s,T\}} \varsigma_v.\rho_v) \tag{5.1}$$

Subject To

$$\sum_{n' \in N'} y^{v}_{\widetilde{n},n'} = 1 \quad \forall v \in \{s,T\},\ \widetilde{n} \in \widetilde{N} \tag{5.2}$$

$$\sum_{n' \in N' : \{n' \neq n\}} y^{v}_{n,n'} \leq 1 \quad \forall v \in \{s,T\}, n \in N' : \{x^{0}_{v,n} = 1\} \tag{5.3}$$

$$\rho_v \geq \sum_{n' \in N' : \{n' \neq n\}} y^{v}_{n,n'} \quad \forall v \in \{s,T\}, n \in N' : \{x^{0}_{v,n} = 1\} \tag{5.4}$$

$$x^{1}_{v,n'} = y^{v}_{n,n'} + x^{0}_{v,n'}.(1 - \rho_v) \quad \forall v \in \{s,T\},\ n \in N,\ n' \in N' \tag{5.5}$$

$$\sum_{n' \in N'} x^{1}_{v,n'} = 1 \quad \forall v \in \{s,T\} \tag{5.6}$$

$$\sum_{v \in \{s,T\}} x^{1}_{v,n'} \leq 1 \quad \forall n' \in N' \tag{5.7}$$

$$r_n = c_n + (\sum_{v \in \{s,T\}} \sum_{n' \in N'} y^{v}_{n,n'}.c'_v) \quad \forall n \in N' \tag{5.8}$$

$$\sum_{v \in \{s,T\}} x^{1}_{v,n}.c'_v \leq r_n \quad \forall n \in N' \tag{5.9}$$

$$\sum_{j:(i,j) \in L} q^{v}_{i,j} - \sum_{j:(j,i) \in L} q^{v}_{j,i} = x^{1}_{v,i} - x^{1}_{s,i}\ \forall i \in N',\ v \in T \tag{5.10}$$

$$\sum_{i \in S} \sum_{j \in S} \sum_{v \in T} q^{v}_{i,j} \leq |S| - 1 \quad \forall S \subset N', 2 \leq |S| \leq N' \tag{5.11}$$

$$\sum_{(i,j) \in L} q^{v}_{i,j} \leq \gamma \quad \forall v \in T \tag{5.12}$$

$$\theta_{min} \leq \sum_{(i,j) \in L} q^{v}_{i,j} \leq \theta_{max} \quad \forall v \in T \tag{5.13}$$

$$\theta_{max} - \theta_{min} \leq \delta \quad \forall v \in T \tag{5.14}$$

$$z_{i,j} \geq q^{v}_{i,j} \quad \forall v \in T, (i,j) \in L \tag{5.15}$$

$$z_{i,j} b' - t_{i,j} \leq t'_{i,j} \quad \forall (i,j) \in L \tag{5.16}$$

$$t'_{i,j} \leq b_l \quad \forall e : (i,j) \in L \tag{5.17}$$

First, we start by repairing all failed VMs using Constraint (5.2). Note that this constraint is only used in the case of source or recipient node(s) failures. Whereas in the event of a

substrate link or a network node failure, this constraint will be omitted since disconnected receivers do not necessarily need to be relocated. Next, intact (active) virtual nodes are allowed to migrate to new substrate nodes in the service of making a better restoration solution. As we have previously mentioned, migration enlarges the search-space and allows to explore more feasible solutions that can yield a lowest-cost tree, and further allows to avoid cases where fixing the location of the active virtual nodes yields no feasible restoration solutions. Active virtual nodes migration is achieved with Constraints (5.3) and (5.4). Constraint (5.5) is used to indicate the new node mapping solution. Observe that the new node mapping solution ensures that every virtual node in a particular MVN request is mapped on a distinct substrate node in order to reduce the impact of failures via Constraints (5.6) and (5.7). Further, it is imperative to ensure that the new node mapping solution respects the substrate network's capacity constraints. Hence, Constraint (5.8) represents this post-migration resources release. While, Constraint (5.9) ensures that the new node mapping solution does not violate the residual capacity of the substrate nodes. To complement the node mapping solution, a link mapping solution must be constructed to route the traffic between the relocated receivers and the multicast source.

Subsequently, the flow conservation constraint (Constraint (5.10)) allows to reconstruct the multicast tree, and Constraint (5.11) ensures that the constructed tree is cycle-free (subtour elimination constraint). Next, the multicast tree maintenance constraint guarantees that the newly constructed multicast tree satisfies the end-to-end delay via Constraint (5.12), as well as the delay-variation (Constraints (5.13) and (5.14)).

Finally, Constraint (5.15) indicates the set of substrate links that compose the multicast tree, while Constraint (5.16) measures the traffic provisioned on each link, and Constraint (5.17) ensures that these latter do not violate the substrate links's capacity.

5.4.3 Complexity Analysis

When a facility node fails and affects a hosted MVN G^v, the MVN Restoration (MVNR) problem requires finding another feasible facility node (with sufficient resources) to host the failed VM, and can connect to the rest of the multicast group with the lowest-cost delay-constrained tree. Here, we consider the cost[1] as the total bandwidth consumed by the restored tree; thus normalized by the bandwidth demand b, the cost becomes the number of substrate links used. We represent the hosts of the unaffected VMs as K, and the set of feasible facility nodes that can accommodate the failed VM as Q; the MVNR decision

[1] A penalty can be added to account for service disruption.

problem can be formulated as follows:

Problem Definition 5.1. *Given a substrate network $G^s = (N, L)$; where N'=N-1 denotes post-failure active substrate node, an MVN $G^v = (V, b, \gamma, \delta,)$, a failed VM $\hat{v} \in V$, a set $K \sqsubseteq N'$ indicating the hosts of the unaffected VMs ($\{V\text{-}\hat{v}\}$), and a set $Q \sqsubseteq \{N'\text{-}K\}$ representing the feasible new locations/hosts for $\hat{v}$; is there a host $x \in Q$ such that $K \cup \{x\}$ can be connected by a tree in G^s satisfying the delay constraints δ and γ, and has at most μ links?*

Theorem 5.1. *The MVN restoration problem is NP-Complete.*

Proof. MVNR can be easily seen in the NP-class; since given a restoration solution, it can be verified in polynomial time. Next, we will prove that the problem is NP-Complete via a reduction from the NP-Complete graph-based Steiner Tree (ST) problem [122]. Throughout the proof, we consider that the failed facility node is omitted/removed from the network. For the sake of completeness, we provide below a definition of the ST problem:

Problem Definition 5.2. *Given an undirected weighted graph G=(N, E) and a subset of nodes $R \sqsubseteq N$; is there a tree connecting R with a cost less than or equal to w?*

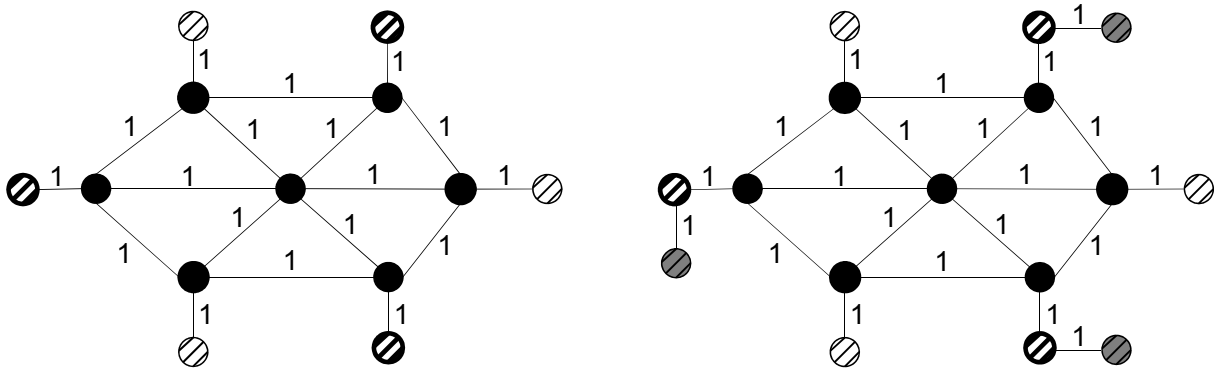


Figure 5.3: Reduction from the graph-based Steiner Tree Problem

Note that ST is NP-Complete even when the weight on all the edges is uniform. Now, given an instance $(G = (N, E), R, w)$ of the ST problem, we transform it in polynomial time into an instance of the MVNR problem as follows: we build a substrate network $G^s = (N', E')$ by adding to G a set of auxiliary nodes $R' = |R|$ (with capacity set as a very large number $\mathcal{M}$); where each node in R' is connected to a single distinct node in R via a single auxiliary link of weight 1. Furthermore, we create an MVN $G^v = (V, b, \gamma, \delta)$ with $|V| = |R| + 1$, assign $\hat{v}$ to be an arbitrary node in V, $K = R$ and $Q \sqsubseteq \{N' - K\}$, $\gamma = \delta = \infty$, and $\mu = w + 1$.

Figure 5.3 illustrates an example of our reduction; the highlighted vertices (in Figure5.3(a)) represent R, and the grey ones (in Figure5.3(b)) indicate the set of auxiliary vertices R'. Clearly, this transformation can be done in polynomial-time. We now show that G has a Steiner tree connecting the vertices in R of cost $\leq w$, $\iff$ $\exists$ a host $x \in R'$ such that $R \cup \{x\}$ can be connected by a tree in G^s with at most $w + 1$ links. If ST has a solution P of cost $p \leq w$, then we can map the failed node $\hat{v}$ on any node $x \in R'$ and augment the tree P with the auxiliary link from x to its corresponding node in R to obtain an MVN restoration solution of cost $p+1 \leq w + 1$. Conversely, if MVNR has a solution of cost $p' \leq w+1$, since $\hat{v}$ may be mapped on a node $x \in R'$, by removing the link from x to its corresponding node in R, we obtain a tree linking the nodes in R of cost p'-1 $\leq w$, that is, a solution to ST in G of cost $\leq w$. This completes the proof of the reduction, we conclude that the MVNR problem is NP-Complete. □

Note that by restriction, MVNR remains NP-Complete for any other instance with strict delay constraints.

5.5 REAL: A Restoration Algorithm for Multi-Rooted Tree Data Center Network Topologies

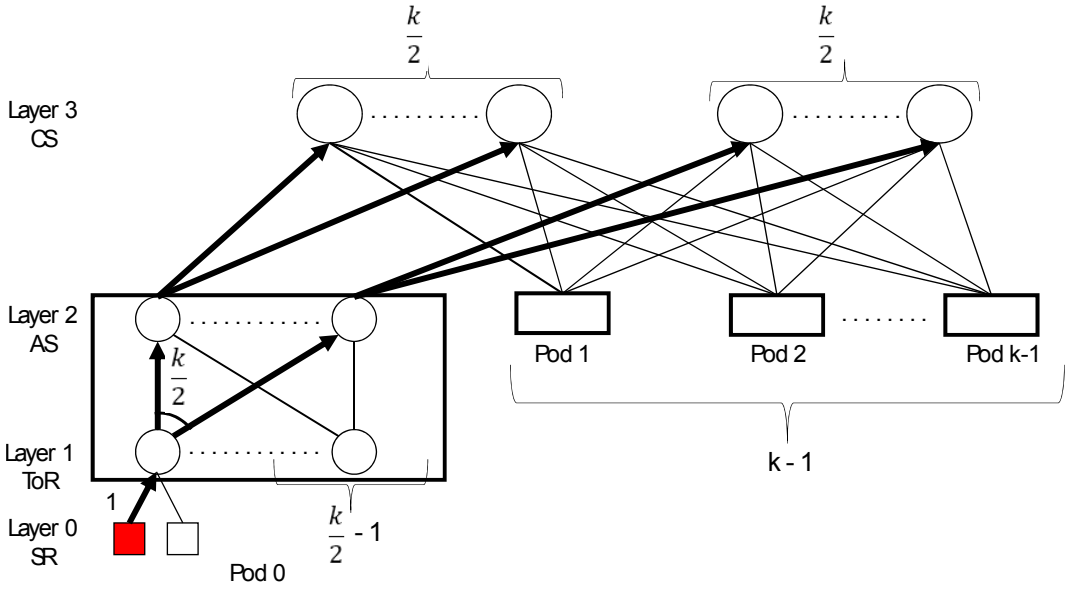


Figure 5.4: FatTree Network

Most data center networks today adopt a multi-rooted tree topology [72, 78], which consists of several layers of commodity switches used to interconnect a large number of servers via multiple equal-cost paths. The goal of this design is to provide a high bisection bandwidth, and eliminate network oversubscription [78]. Figure 5.4 illustrates an example of such network topologies known as the FatTree network. The FatTree network consists of a multi-rooted tree-like topology built out of 3 layers of k-port switches, that interconnect $\frac{k^3}{4}$ Server Racks (SR). It consists of k pods, where each pod hosts 2 layers of switches: $\frac{k}{2}$ Aggregate Switches (AS) connected to $\frac{k}{2}$ Top of Rack (ToR) switches, forming a complete bipartite graph inside every pod. Further, each $\frac{k}{2}$ ToR is connected to $\frac{k}{2}$ SRs, and each $\frac{k}{2}$ ASs is connected to $\frac{k}{2}$ Core Switches (CS). It has been shown in many cases that these data center network topologies can be leveraged to enhance the support of multicast in data centers [71, 78, 79]. We add to these findings by proposing REAL: a novel restoration algorithm that can find the optimal multicast restoration solution in a multi-rooted tree like network topology in polynomial time.

In order to ensure that our restored multicast tree is always loop-free, our restoration lookup allows the traffic in the resultant tree to change its direction once; that is up-packets can change their direction once to move traffic downwards. However, a downward facing packet can never be re-directed upwards. This aforementioned constraint ensures that the resultant tree will never have any cycles [19].

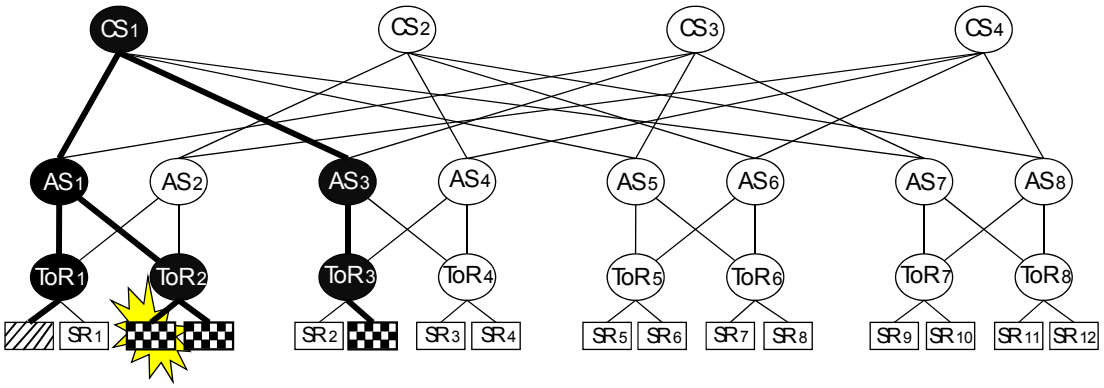


Figure 5.5: FatTree Network

Theorem 5.2. *The optimal multicast restoration problem can be solved in polynomial time in the FatTree network.*

Proof. **Case of Recipient Node Failure:** Our algorithm begins by pruning the substrate network in order to remove server racks and/or substrate links that violate the resource

demands of the failed receiver. Next, we search via the remaining components in the MVN tree to find the SR that can host the failed receiver while incurring the minimal amount of additional Steiner points. To do so, we perform a hop-to-hop search from each active VM in the tree. Further, the look-up is encouraged to go through existing links in the multicast tree by setting the weight on these links to 0. Subsequently, the first feasible SR that will be reached by any VM will yield the lowest cost restoration solution, since it will incur the minimal number of additional Steiner points. Indeed, the set of candidate hosts that will be reached first are the SRs that share the same ToR with active VMs. If any of these candidate hosts was deemed feasible (in terms of resource capacity and respecting the delay constraints), then this solution will yield 0 additional Steiner points. If none of the candidate hosts was deemed feasible, the search will persist by exploring the ASs, yielding at most 2 additional Steiner points; and finally via the CSs. Clearly, If no solution can be found, then the MVN with the current embedding solution cannot be restored. Alternatively, the first solution found will definitely yield the lowest cost restoration solution. The topological structure of the FatTree network provides every pair of servers with $\frac{k^2}{4}$ equal cost paths. Further, there can be no more than $\frac{k^3}{4}$ candidate SRs. Hence, if we restrict the search from the source node only, we have a worst-case complexity of O(k^5), which is indeed polynomial in the size of the substrate network. To better illustrate this, consider the example in Figure 5.5, where a 3-receiver MVN is hosted in a FatTree (k=4). Given a recipient node failure, we launch the hop-to-hop search from the SRs in the current MVN. Here, we find that SR_1 and SR_2 are reachable from the ToRs in the current tree (ToR_1 and ToR_2). Note that choosing either one of these SRs will yield the lowest cost restoration tree. However, it could happen neither SR_1 nor SR_2 are feasible (e.g., due to resource scarcity), then the hop-to-hop search will continue by moving towards the ASs and so on, until a feasible solution is found.

Case of Source Node Failure : Now, the problem becomes that of finding a SR that can host the source and yield the lowest cost delay-constrained Steiner tree. Here, we prove that by adopting a path-convergence approach from the recipient nodes, the first feasible SR that these nodes will converge to will definitely yield the lowest cost restoration solution. We distinguish between two cases: Case 1, where the recipient nodes are residing in the same pod, and Case 2 where the recipient nodes are distributed across multiple pods. In case 1, if the recipient nodes are sharing the same ToR, then they will always converge first towards any feasible SR connected to that same ToR. If no solution was found, the next convergence will occur at the intra-pod SRs. Finally, in the event where no intra-pod solution can be found, then the next convergence will occur at SRs residing in alien pods. Clearly, the first

solution found will yield the lowest cost restoration tree, as it will be composed of fewer Steiner points interconnected via lower-cost (lower-layer) substrate links. Now in case 2, all receivers will explore all SRs at the same time; and the algorithm will return the lowest-cost solution found. Given that to each candidate source node, we can find at most $\frac{k^{2|T|}}{4}$ distribution trees; hence we have a worst-case complexity of $O(k^{2|T|+3})$, which is polynomial in the size of the network. □

5.6 Numerical Results

We evaluate how well our proposed restoration schemes perform. We compare our proposed restoration algorithm REAL against the restoration model proposed in Section 5.4.2, and two benchmark algorithms: First-Fit node embedding with Steiner tree reconstruction (denoted as Steiner), and a Greedy node mapping with shortest path restoration (denoted as Greedy). Here, we distinguish between two variations of our proposed model: migration-aware restoration model, and no-migration restoration model that is obtained by omitting Equations (5.3), (5.4), and (5.8). Hence, the new node mapping solution obtained via Equation 5.5 will maintain the old node mapping solution of all intact virtual nodes. Further, we slightly modify Equation 5.9 to perform substrate nodes capacity check for the new hosts of the failed virtual nodes only. Throughout our numerical analysis we adopt the no-migration variation of REM.

The Steiner restoration scheme consists of finding the first host that can accommodate the failed node, tear down the initial tree and then reconstruct the Steiner tree to connect the intact VMs with the new host. The greedy restoration on the other hand consists of sorting the hosts based on their residual resource capacity, and iteratively select the host with the highest capacity to host the failed VM. If the failed VM is a recipient node, then the shortest path from the new host to the MVN source node is established (using Dijkstra); alternatively, if the failed node is the source, then the shortest path from the source to each receiver is established (in a source-driven fashion).

To perform this evaluation, we look at three main metrics: execution time, restoration ratio (RR), and total revenue achieved; where restoration ratio measures the percentage of restored MVNs out of the total number of failed MVNs. Throughout all our simulation, we set the capacity of the substrate nodes to 64 GB, and that of substrate links to 1 Gbps. Further, all our MVNs are randomly generated with size ranging between [2-14] receivers over FatTree (k=4), and [5-30] receivers over FatTree (k = 8) and (k=16) and the two random networks.

The resource demand of the virtual nodes and links are randomly selected from the range between [2-12] and [50-300], respectively. We assume that the VNs arrival follows a poisson process with an arrival rate λ per time unit, and the departure follows a negative exponential distribution with a service rate μ per time unit. We assume that the VNs arrival follows a Poisson process distribution with a normalized load of 10, and are embedded using one of our proposed embedding techniques in Chapter 3. To simulate failures, we setup a random failure generator with a frequency metric F, where F defines the interval of random failures that will occur over time, and we set a constant Mean Time To Repair ($MTTR$) interval for the failed nodes.

Table 5.2: Average Execution Time (ms) - Restoration over FatTree Network

	REM	REAL	Steiner	Greedy
FT (K=4)	417	2	2	2
FT (K=8)	540745	57.25	12.75	8.5
FT (K=16)		123	132	60.25

a) **Execution Time:**

First, we look at the execution time achieved by each of the aforementioned techniques. As we are dealing with a reactive scheme, execution time is of paramount importance as it is additive to the restoration time, and therefore greatly impacts the downtime or slow-time experienced by the service during restoration. Table 5.2 illustrates the average execution time obtained over FatTree (k=4), (k=8), and (k=16). Clearly, we observe that the runtime of the REM grows exponentially as we move from FatTree (k=4) to (k=8) reaching an average of 9 minutes to compute a restoration solution for a single MVN. On the other hand, we observe that our restoration algorithm, as well as the Steiner and Greedy remains in the order of milliseconds on average.

b) **Restoration Ratio:**

Second, we look at the restoration ratio, since a fast execution time is only significant if it returns good quality solutions. Figures 6(a)-6(c) illustrate the obtained results over FatTree (k=4), (k=8), and (k=16), respectively. Over FatTree (k=4) (Figure 6(a)), we observe that both REM, as well as REAL achieve 100% restorability as we vary the failure frequency from 2 to 8. On the other hand, the restorability of both the Steiner and Greedy restoration schemes are greatly affected as we increase the failure frequency. Indeed, we observe that as the failure frequency hits 8, the restoration ratio of Steiner drops to 60%,

and that of Greedy to 46%. Similar results are observed over FatTree(k=8) (Figure 6(b)) and FatTree (k=16) (Figure 6(c)). Again, we observe on FatTree (k=8) that REAL achieves a 100-95% restoration ratio, whereas Greedy's restorability drops from 100% to 58%, and Steiner's from 100% to 61%. Overall, we can conclude that REAL achieves equal restorability to the restoration model, and a much higher restoration ratio than both Greedy and Steiner. Further, the Steiner restoration scheme achieves better restorability than Greedy. Greedy's source-driven tree construction yield significant redundant traffic and poor resource utilization, thereby greatly impacting its restorability as the failure frequency increases. On the other hand, building the lowest-cost tree (Steiner), does not always necessarily guarantee that the latter will satisfy the delay-constraints of the MVN in question, hence Steiner fails to find feasible restoration solutions due to its limited search space.

c) **Total Revenue:**

Finally, we look at how the restoration capability of each of the aforementioned schemes affects the long-term revenue achieved by the cloud provider. We measure the total revenue using the following equation:

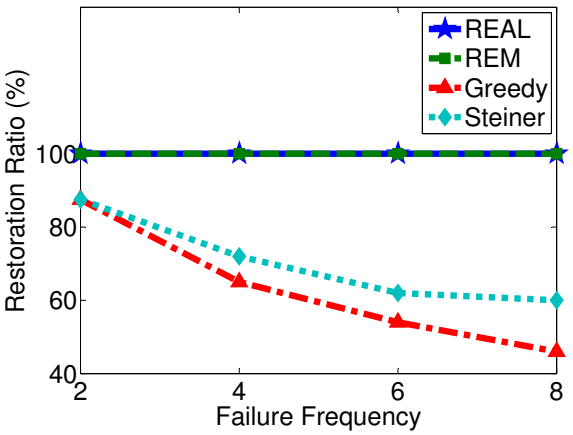
$$Revenue = \sum_{v \in \{s,T\}} d_v + \sum_{e' \in E'} b'. \tag{5.18}$$

Further, whenever the adopted method fails to restore a failed MVN, then the cloud provider will be penalized by remitting 50% of the revenue gained from hosting this particular service. Figures 6(d)-6(f) show our obtained results over FatTree (k=4), (k=8), and (k=16). Here we draw upon two main observations: First, all adopted restoration techniques experience a revenue drop as we increase the failure frequency, since the failed substrate nodes shrink the network's admissibility. Second, we observe that the low restorability exhibited by both Steiner and Greedy leads to a much lower achievable revenue. Indeed for a failure frequency of 8, REAL achieves 10-15% higher revenue than Greedy and Steiner over all tested network topologies.

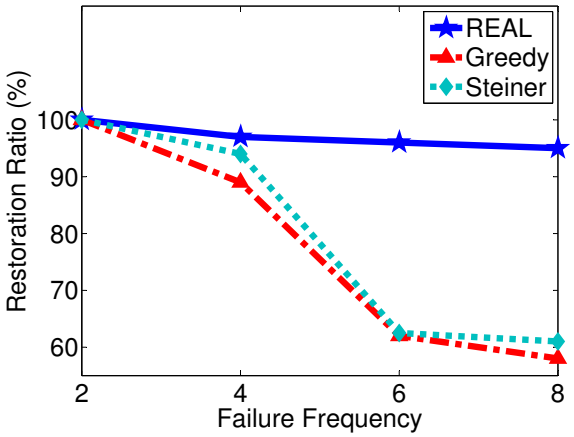
5.7 Conclusion

In the proposed manuscript, we studied the impact of failure on MVNs hosted in data center networks. We mathematically formulated the migration-aware post-failure restoration of MVNs, and proved its NP-Complete nature in arbitrary graphs. Further, we exploited the

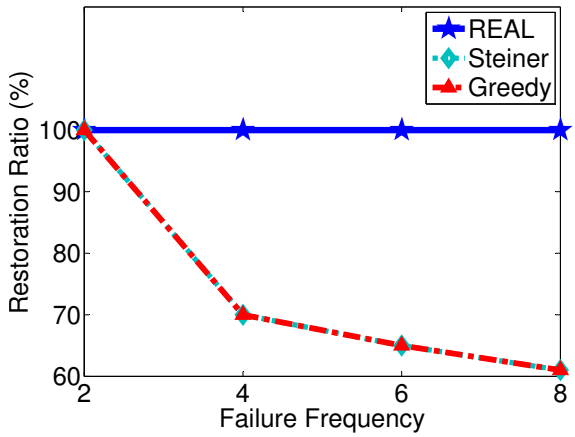
structured topology of tree-like data center networks to prove that the MVN restoration problem can be solved to optimality in polynomial time. Our numerical results covenant for the promising restorability achieved by our proposed scheme within considerably fast execution time.



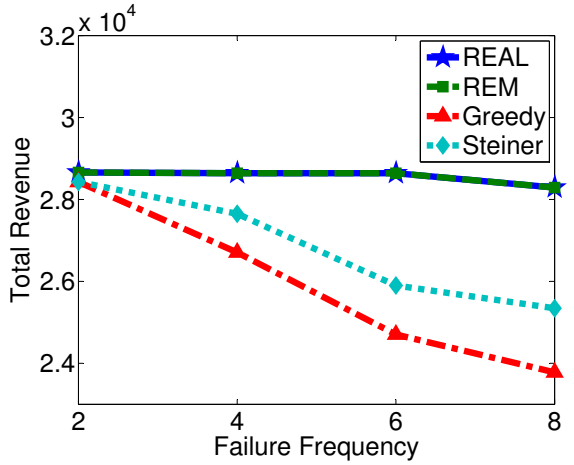
(a) RR - FT ($k = 4$)



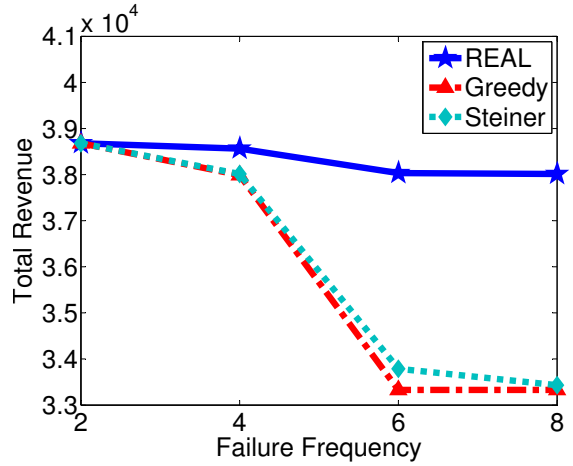
(b) RR - FT ($k = 8$)



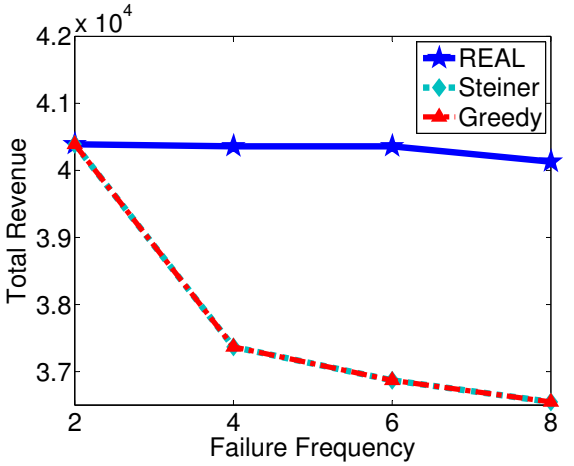
(c) RR - FT ($k = 16$)



(d) Revenue - FT ($k = 4$)



(e) Revenue - FT ($k = 8$)



(f) Revenue - FT ($k = 16$)

Figure 5.6: Performance Analysis

Chapter 6

A Reliable Embedding Framework for Elastic Virtualized Services in the Cloud

6.1 Problem Statement

In Chapters 4 and 5, we considered the problem of the survivability of unicast and multicast VNs against facility node failures. These fail-over techniques guarantee 100% availability of cloud services against any facility node failure. Such a scheme is particularly attractive for services with mission-critical components that do not tolerate any failure. Another potential fail-over mechanism is to provide an availability guarantee for hosted services. Service availability guarantees a ratio of uptime over the total elapsed time for the tenant's service; e.g., a five 9s availability guarantees less than 6 minutes service downtime per year.
Providing availability guarantees to tenants is commonly referred to as the availability-aware VNE [58, 59]. Here, cloud providers must allocate resources to tenant's services; while making sure that the availability of the physical hardware running the tenant's service satisfy the availability guarantee requested by the tenant. This problem is particularly challenging given that data center hardware exhibit heterogeneous failure rates. The heterogeneous environment of cloud infrastructures is due to the fact that cloud hardware are not provisioned at the same time; rather cloud providers acquire resources gradually to grow their infrastructure [123]. This incremental provisioning process leads to having different generation hardware from different vendors and specifications cohabiting the same infrastructure. This aspect has been pointed out several times in the literature [104, 123–125] based on real data center traces from VMware [123], Microsoft [104], IBM [126], and Google [125]. In fact, according to Microsoft [104] servers are meant to last as long as 3 to 5 years, with increments

ordered quarterly or yearly. The heterogeneous configuration of machines has been further elucidated in [125]; where a table characterizing the varying platform and memory/compute ratio of available machines in the cluster is presented.

A handful of contributions [58,59] recently emerged to propose availability-aware embedding techniques. The two main limitations of the aforementioned work is that they have overlooked the problem of "availability-overprovisioning"; that is providing services with more availability guarantees than requested. As this chapter will show, such a scheme leads to poor resource utilization, thereby decreases network admissibility. The second limitation is the assumption that cloud services's resource demands remain static throughout their residency. When in fact network load may change over time; this is particularly true for services with periodic resource demands (e.g., an online shopping store during holiday seasons, streaming election results, etc.). Elasticity is notably one of the most attractive features of the IaaS paradigm [127,128]. This flexibility in scaling up and down the provisioned resources allows to swiftly adapt to services' actual needs over time. Indeed, it has been shown that 90% of IT services exhibit periodic resource demands [60], hence SPs/tenants can decide to increase their leased computing resources to meet the peaks, and decrease them during steady or idle times to economize. In fact, cloud providers are aware of this common demand pattern, therefore a number of them (e.g, Amazon EC2 Cloud, GoGrid) offer different type of instances to encourage tenants wishing to host applications with unpredictable workload to opt out from reserving steady resources.

While managing a scale-up request, the current network elements hosting the scaling service may fail to meet the requested changes. Thus, more resources may need to be provisioned. For example, the service may request additional network bandwidth to mitigate network delays and/or reduce completion times, or virtual machines need more computing resources, or the level of availability provided by the current hosting elements may not be satisfactory. In response, the cloud provider must make the necessary reconfigurations/adjustments, all while making sure that the requested level of availability is maintained. As this manuscript will show, the problem of managing scaling requests for VNs with strict availability demands is not intuitive, and encompasses multiple challenges. Hence, the cloud provider must weigh-in these various inter-playing factors and make the rightful decision that best serves his/her design objective (e.g., maximize the long-term revenue). To the best of our knowledge, the problem of managing reliable elastic[1] VNs has not been considered in prior literature.

To this extent, we propose RELIEF: a reliable embedding framework for elastic VNs in

[1]Throughout this chapter we use the terms elastic and scaling interchangeably.

failure-prone data center networks. It consists of two main modules: an availability-aware embedding module for incoming VN requests, and a reliable reconfiguration module to manage scaling requests of hosted services. As opposed to existing work [58, 59] on availability-aware embedding, our proposed scheme achieves resource utilization efficiency by providing "just-enough" availability. This allows to circumvent resource wastage incurred by availability over-provisioning. Further, we introduce the concept of *protection-domains* to equip our proposed scheme with the ability to augment services with redundant nodes to enhance their availability. This novel concept is also useful when managing scaling requests of services with low tolerance to disruption. Hence, our main contributions can be summarized as follows:

- We formally define the availability-aware resource allocation problem and prove its NP-Hard nature.
- We introduce the concept of "protection-domains" to alleviate availability breaches, and we propose JENA: a novel Tabu-based search to perform VN placement with Just-ENough Availability guarantees.
- We formally define the problem of managing VNs scaling requests in failure-prone data centers, and we highlight the tradeoff between a migration and a redundancy enabled scheme.
- We propose ARES: a novel Availability-aware, migration and redundancy enabled, REconfiguration Scheme to manage VN scaling requests.
- We evaluate the performance of our proposed modules against peer techniques extracted from the literature, as well as benchmark algorithms, and we show that our framework outperforms its peers in terms of network admissibility, and achievable revenue.

6.2 Related Work

6.2.1 Availability-Aware VNE

Given the failure-prone nature of data center networks, significant effort [48–55, 58, 59, 129–131] has been devoted to solve the resource allocation problem in the cloud with survivability guarantees. The aforementioned work consists of providing 100% availability guarantees under various conditions; for instance some [53–55, 129, 131] considered survivability against

single facility node, while others accounted for the case of [48, 50, 131] single link failure, or considered [49, 51, 52] the problem of survivability against risk-group failures. Another wave of work [130,132,133] focused on providing fault-tolerance guarantees. For instance, the work in [130] proposed to enhance the survivability of VNs by spreading their VMs across multiple fault-domain, such that a "worst-case survival" guarantee can be achieved while minimizing bandwidth consumption. Similarly, the work in [132] characterized fault-domains in data centers, and provided a correlation study between recovery time and failure complexities, showing that when a failure affects more than one service component, the recovery time of the latter significantly increases.

Another fail-over mechanism that emerged in the literature is the VNE problem with availability guarantees [58, 59]. In [58], the authors proposed a novel technique to compute VNs availability by listing all possible failure scenarios, and factoring-in their conditional probability. Their proposed embedding technique consists of a greedy embedding where all possible mapping solutions are explored. It is important to note that this former work considers replication groups (redundant/backup nodes) to be given. Whereas the work in [59] considered the problem of embedding VN requests with a star topology, and also proposed a greedy availability-aware embedding technique. Here, both the availability of facility nodes and network nodes are considered, and VMs of the same VN are allowed to be collocated on the same physical machine. In the event where a VN embedding fails to satisfy the requested availability, the latter will be augmented with k backup nodes, where k is equal to the minimum number of collocated VMs, which may lead to significant backup footprints.

While VM collocation can indeed reduce bandwidth consumption, it can also severely impact the fault-tolerance (worst-case survival guarantee) of the VNs in question, not to mention the increase in recovery time as shown in [132], and idle resources for provisioned backup/redundant resources. When all the VMs of a particular service are placed on a single facility node, then a single failure will bring down the entire VN; whereas when the VMs are spread across multiple servers (fault-domains), the failure of a particular facility node (or a subset of nodes) does not necessarily bring down the entire service, it may still remain operational only with a degraded performance [130].

Our work is different, as we propose a reliable embedding framework that also manages VN scaling requests over time. Further, as opposed to existing work, our availability-aware embedding module is topology-independent, and aims to provide just-enough availability-guarantees. This former attribute leverages the utilization efficiency of the substrate resources, thereby rendering higher admissibility than existing work [58].

6.2.2 Elastic Services in the Cloud

When a service's demands changes over time, the initial facility nodes hosting this service may not have sufficient residual capacity to meet the requested changes [134]. In this regard, numerous work [121, 135–138] have been devoted towards managing scaling requests. In [135], the authors proposed a multi-agent learning algorithm introduced at every substrate node and link. The role of these agents is to monitor the actual substrate network usage by each hosted VN and dynamically re-adjust (reconfigure) the amount of reserved resources for each VN depending on the perceived needs. The work in [136] considered the problem of embedding VNs with periodic resource demands. The authors presented two embedding techniques: one that assumes that the periodic demands of any VN is known apriori, and another that predicts the resource demand of VNs based on their historic demand pattern, and adjusts the amount of allocated resources to each VN accordingly. In [137], the authors proposed an incremental re-embedding scheme for evolving VN requests that considers both resource demand increase, as well as new components arrival. Their incremental process consists of tackling resource demand increase first, then network components arrival. The authors also employed resource migration to support resource demand increase. However, it is important to note that such an approach is only applicable for horizontally scalable services (e.g., distributed storage system). Similarly, the work in [138] also addressed the problem of reconfiguring existing VN requests as they evolve with time. The authors considered 4 scenarios of VN request changes: increase/decrease of resource demands, and arrival/departure of new network components. They proposed two heuristic algorithms to handle the increase of resource demands and new components arrival. However, their proposed method exhaustively considers all mapping combinations to find the most cost-efficient one. Hence, their proposed method is not suitable when employed over large substrate networks. Finally, the work in [121] proposed a migration-aware embedding and resource consolidation framework to achieve energy conservation. Their framework only considers resource demand increase, and consists of a greedy embedding that handles scale-up requests by evaluating the cost of each candidate host, to find the lowest cost reconfiguration solution.

Our work distinguishes itself from the above-surveyed literature in that it addresses the problem of managing availability-aware elastic VNs. In this context, the management of elastic VN requests requires the reconfiguration of the VN's embedding solution while also preserving its availability. To the best of our knowledge, this problem has not been tackled in the literature before. Hence, this chapter is devoted towards studying this problem, and proposing a novel framework that performs reliable embedding and reconfiguration of elastic

services in the cloud.

6.3 RELIEF Framework Overview

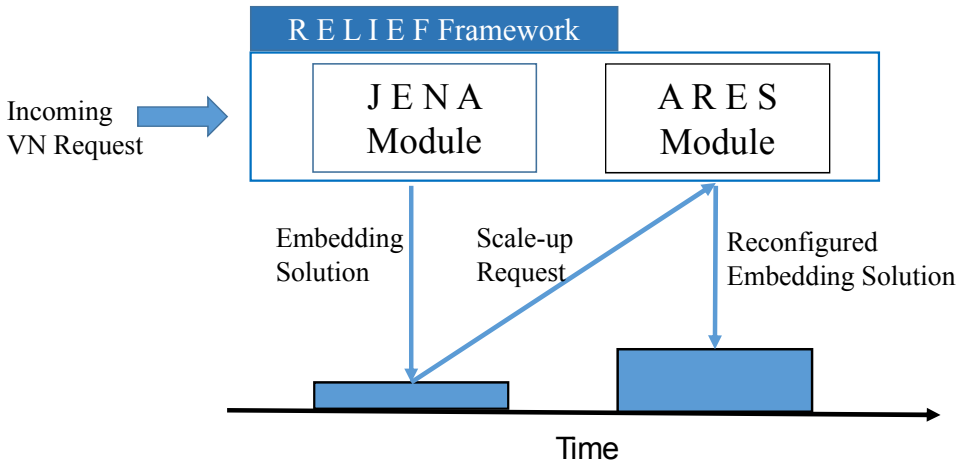
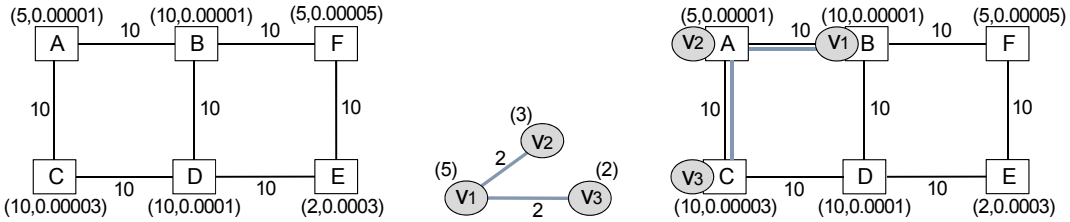


Figure 6.1: RELIEF Framework

First, we begin by giving an overview of our proposed framework. RELIEF is an availability-aware embedding and reconfiguration framework for elastic services in the cloud. It encompasses two main modules (as shown in Figure 6.1): JENA, the VN Embedding (VNE) module with Just ENough Availability, and ARES, the Availability-Aware REconfiguration Scheme. Our proposed framework is performed online, upon the arrival of any VN request. It begins by processing an incoming VN request by invoking JENA to return an embedding solution that meets the requested availability. During the residency time of the VN, its request may evolve/change overtime; these scale-up (or down) requests are treated by ARES to reconfigure the embedding solution accordingly to meet the requested changes, while maintaining the availability of the VN in question. Throughout the embedding and reconfiguration, RELIEF aims to provide the hosted services with the requested resource demands, while minimizing the amount of squandered availability. As opposed to existing work [58, 59], our proposed work aims to achieve efficient use of network resources by providing "just-enough" availability, which in turn yields a much higher admissibility (as shown in our numerical evaluation). In the remaining of the manuscript, we thoroughly revise the problem behind each proposed module.



(a) Substrate (b) VN (c) Availability-Aware VNE

Figure 6.2: Network Model

6.4 Availability-Aware VNE problem

1. **The Substrate Network :** We represent the substrate network as an undirected graph denoted by $G^s = (N,L)$, where N is the set of substrate nodes, and L is the set of substrate links. Each substrate node $n \in N$ is associated with R resource types (CPU, memory, etc.), each with a finite capacity denoted by c_n^r. Moreover, each substrate node is associated with a unique failure rate, denoted as f_n, which indicates the future probability that substrate node n will fail based on its historic failure pattern. Similarly, each substrate link $l \in L$ has a finite bandwidth capacity, denoted by b_l.
   Figure 2(a) illustrates a substrate network with 6 nodes. Throughout our motivation examples we present a simplified representation of a substrate network by only showing the facility nodes interconnect. It is important however that throughout our numerical analysis we adopt realistic networks with servers interconnected via several layers of network nodes (routers/switches). Further, for the sake of clarity we show a single resource type (represented by the number in parenthesis above each node), followed by the node's failure rate. Similarly, we observe that the substrate links interconnecting the network nodes exhibit 10 units of bandwidth capacity each (represented by the number next to each substrate link). To obtain the failure rate of a particular node, let a_n denote the availability of substrate node n; then $f_n = 1 - a_n$, where the availability of n can be computed as follows:

$$a_n = \frac{MTBF_n}{MTBF_n + MTTR_n} \tag{6.1}$$

   $MTBF_n$ and $MTTR_n$ denote the mean time between failure and mean time to repair for substrate node n.

2. **The Virtual Network (VN) :** We represent a VN as a set of virtual nodes (virtual

machines), interconnected via virtual links. The virtual links correspond to the communication requirements between the virtual nodes in a given VN request. We denote a VN as a graph $G^v = (V,E)$, where V represents the set of virtual nodes, each with resource demand of $c_v'^r$ for each resource type $r \in R$ (CPU, memory, etc.). E represent the set of virtual links, where $o(e)$, $d(e)$, and b'_e denote the origin, destination, and bandwidth requirement of each $e \in E$, respectively. Figure 2(b) shows an example of a VN request with 3 virtual machines (nodes) interconnected via two virtual links. In addition to the resource and bandwidth demands of the virtual nodes and links, each incoming VN request (tenant) demands a specific availability requirement A_{req} for its service (in this example, we assume that $A_{req} = 99.95\%$).

The availability-aware VNE problem is defined as:

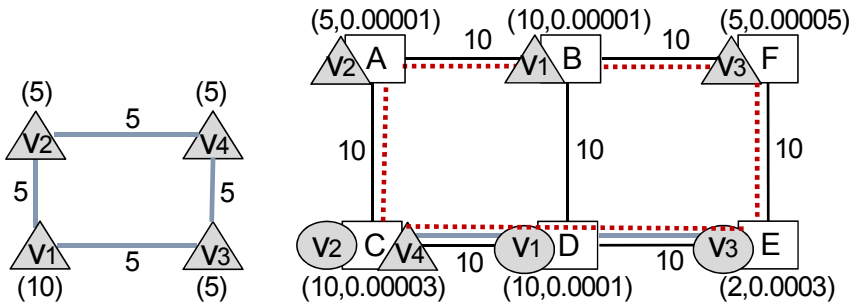
Problem Definition 6.1. *Given a substrate network G^s, and a virtual network request G^v; find the minimum cost mapping solution of G^v onto the substrate network, such that the overall availability requirement A_{req} of the VN request is satisfied, as well as its resource demands, without violating the capacity constraints of the substrate network.*

Similarly to the VNE problem, the availability-aware VNE problem can also be decomposed into two subproblems: the VMs Placement (VMP), and the Virtual Links Routing (VLR) subproblems. Let s represent a feasible mapping $s = (s_N,s_E)$ of a given VN request, which holds the solution for the two subproblems:

- VMP: s_N: $v \longrightarrow N$.
- VLR: s_L: $e \longrightarrow P$. P represents the substrate path routing virtual link e.

Note here that by finding the minimal cost mapping solution for each incoming VN request, the infrastructure provider can maximizes the utilization efficiency of the substrate network; which ultimately yield higher admission and long-term revenue. The availability of a VN is depicted by the availability of the facility nodes hosting its VMs. That is, at any given point in time, a VN is said to be available if all of its hosting substrate nodes are up and running. Subsequently, the availability of a VN can be represented as follows[2]: $A = Prob\{\text{each physical server hosting a VM} \in \text{VN is available}\}$, which is computed as the product of the availability of the hosting substrate nodes: $A = \prod_{n \in s_N} a_n$. Figure 2(c) illustrates an example of the VN in Figure 2(b) hosted in the substrate network presented in Figure 2(a). Here, we observe

[2]We disregard network element failures and focus on facility node failures.



(a) VN_2 (b) VNE with "Just-Enough" Availability

Figure 6.3: Greedy Vs. Just-Enough Availability Aware VNE

that v_1 is hosted on substrate node B with failure rate of 0.00001, hence with an availability of 0.99999. Similarly, v_2 is hosted on substrate node A with availability of 0.99999, and v_3 on substrate node C with an availability of 0.99997. This implies that the overall availability of this VN is $A = a_A.a_B.a_C = 0.9999$, that is a $99.99\% \geq A_{req}$ availability catered by the product of the availabilities of the hosting nodes.

Theorem 6.1. *The availability-aware VNE problem is NP-Hard.*

Proof. We prove that the problem at hand is NP-Hard by restriction. Proof by restriction [139] consists of showing that the problem under consideration has at least one specific instance that is known to be NP-Hard. Indeed, the availability-aware VNE problem is equivalent to solving the VNE problem when the availability requirement is 0. Since the latter is also NP-Hard [33], and represents an instance of the availability-aware VNE problem, this leads us to conclude that the availability-aware VNE problem is also NP-Hard. □

6.4.1 Problem Motivation

Given the NP-Hard nature of the availability-aware VNE problem, one possible way to solve it is by performing a greedy embedding. Given a bare-bone[3] VN request and a substrate network, greedy embedding is achieved by first sorting the facility nodes based on their availability measure (in a descending order). Next, the VMs are placed one-by-one, starting with the most available facility node, onwards. Such a greedy embedding will for sure guarantee that the VN's availability requirement will be satisfied if possible; and if no solution can be found, then there is definitely no other subset of substrate nodes that can satisfy the VN's availability demand. While this approach seems reasonable, and has been used in

[3]A virtual network that is not augmented by any backup/redundant node(s) is referred to as a bare-bone network.

the existing literature ([59]), multiple concerns emerge along with it: first, performing a greedy embedding could very much yield to a solution wherein the VN has a much higher availability than requested. This could in turn incur a poor resource utilization, as the capacity of highly available facility nodes will be quickly drained, thereby hindering the admissibility of future VN requests with high availability demand. Hence, the first challenge is finding the embedding solution that can satisfy the VN's resource demand with "just-enough" availability provisioning.

To better illustrate this, consider again Figure 2(c); observe how this solution represents a greedy embedding that surpasses the requested availability of 99.95%; hence yield an excessive availability of 0.04%. Now consider that at time t_1, a new VN request (denoted as VN_2) arrives (illustrated in Figure 3(a)), which demands a high availability requirement of 99.99%. Clearly, there is now no feasible embedding solution for VN_2, since highly available nodes are already occupied, and those remaining fall short from providing the requested availability for this tenant. However, notice that the current embedding for the initial VN (Figure 2(b)) is squandering the substrate network resources by unnecessarily occupying the highly-available nodes; when in fact, hosting v_1, v_2, and v_3 onto substrate nodes D, C, and E (respectively), instead will achieve a "just-enough" availability provisioning of 99.95%, as well as free-up the highly-available nodes, making room for VN_2, and increasing the substrate network's admissibility. Finding the most suitable embedding solution for a given VN, that achieves just-enough availability guarantees, can be a daunting task. In fact, in the case where all substrate nodes have enough capacity to host any virtual node, there are $\frac{N!}{(N-V)!}$ combinations to choose from, which is exponential. Hence, exhaustively enumerating all possible solutions is clearly a bad and expensive proposition.

We build on these observations to propose an embedding approach that achieves just-enough availability, which ultimately distinguishes our work from prior work.

6.4.2 Problem Formulation

In this section, we present a mathematical formulation to the problem of VNE with just-enough availability for bare-bone VNs. The model assumes as input the VN with its availability requirement, as well as the failure rates of the substrate nodes. We set the objective function to minimize the overall computed availability (squandered), under the constraint that the requested availability guarantee is achieved.

- Parameters:

$G^s(N, L)$: Substrate network with N nodes and L links.

$G^v(V,E)$: Virtual network with V virtual nodes and E virtual links.

A_{req}: Requested availability requirement.

- Decision Variables:

$$x_{v,n} = \begin{cases} 1, & \text{if virtual node } v \text{ is mapped on substrate node } s, \\ 0, & \text{otherwise.} \end{cases}$$

$$y^e_{i,j} = \begin{cases} 1, & \text{if virtual link } e \text{ is routed through substrate link } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

A: denotes the computed availability.

$t_{i,j}$: denotes the traffic measured on link (i,j).

- **Mathematical Model:**

$$\text{Min} \quad A$$

Subject to

$$\sum_{n \in N} x_{v,n} = 1 \qquad \forall v \in V \tag{6.2}$$

$$\sum_{v \in V} x_{v,n} \leq 1 \qquad \forall n \in N \tag{6.3}$$

$$\sum_{v \in V} x_{v,n} c'^r_v \leq c^r_n \qquad \forall r \in R, n \in N \tag{6.4}$$

$$A = \prod_v a_v \tag{6.5}$$

$$A \geq A_{req} \tag{6.6}$$

$$\sum_{i:(i,j) \in L} y^e_{i,j} - \sum_{j:(j,i) \in L} y^e_{j,i} = x_{o(e),i} - x_{d(e),i} \qquad \forall i \in N,\ e \in E \tag{6.7}$$

$$t_{i,j} = \sum_{e \in E} y^e_{i,j} b'_e \qquad \forall (i,j) \in L \tag{6.8}$$

$$t_{i,j} \leq b_{i,j} \qquad \forall (i,j) \in L \tag{6.9}$$

Constraints (6.2)-(6.4) perform the VMP subproblem, where Constraint (6.2) indicates that every virtual node $v \in V$ is placed on a substrate node $n \in N$, and Constraint (6.3) ensures

that at most one VM can be placed on a single substrate node. Avoiding conflicted placement of substrate nodes enhances the VN's fault-tolerance [130]. Constraint (6.4) avoids substrate nodes capacity violation. Constraint (6.5) measures the availability induced by the VMP; where $a_v = \sum_{n \in N} x_{v,n} a_n$ denotes the availability of each VM v's host. Constraint (6.6) ensures that the latter respects the requested availability requirement. Constraint (6.7) represents the flow conservation constraint, while Constraint (6.8) measures the traffic routed on each substrate link. Finally, Constraint (6.9) avoids substrate links capacity violation.

Complexity Analysis:

Clearly the presented model is a Mixed Integer Non-Linear Program (MINLP), owing to Equation 6.5. Hence, to solve the model to optimality, we need to linearize it. This can be easily achieved by replacing the product of every two node embedding variables ($x_{v,n} \in \{0,1\}$) with a single variable, and then adding three inequalities to bound the value of the new variable by that of the variables it replaces. However, given N substrate nodes and V virtual nodes, there will be $\frac{N!}{(N-V)!}$ terms to linearize; and for each term, (V-1) additional variables to add, and 3 times more inequality constraints.

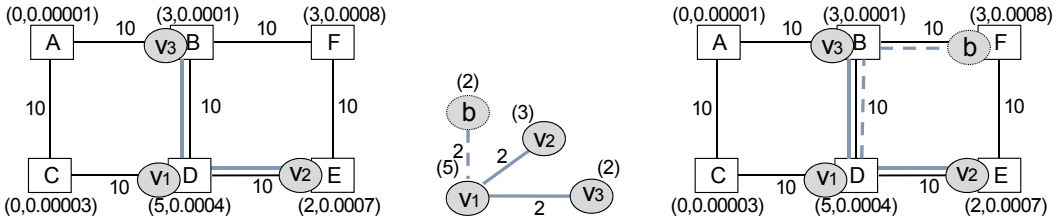
To better illustrate this, consider the availability computation of the bare-bone VN presented in Figure 2(b); further, consider that we are aiming to embed the former on a substrate network with 3 substrate nodes n_1, n_2, and n_3. Subsequently, we have:

$$\begin{aligned} A = (x_{v_1,n_1}.x_{v_2,n_1}.x_{v_3,n_1}).a_{n_1} + (x_{v_1,n_2}.x_{v_2,n_2}.x_{v_3,n_2}).a_{n_2} + \\ (x_{v_1,n_3}.x_{v_2,n_3}.x_{v_3,n_3}).a_{n_3} + (x_{v_1,n_1}.x_{v_2,n_1}.x_{v_3,n_2}).a_{n_1}.a_{n_2} + \\ (x_{v_1,n_1}.x_{v_2,n_2}.x_{v_3,n_3}).a_{n_1}.a_{n_2}.a_{n_3} + ... \end{aligned} \tag{6.10}$$

Hence to linearize the first term in the equation, we can replace $x_{v_1,n_1}.x_{v_2,n_1}$ with z_1, where the first term becomes $z_1.x_{v_3,n_1}$, which in turn can be replaced with z_2. The value of z_1 is a function of the values of x_{v_1,n_1} and x_{v_2,n_1}, which can be expressed as follows:

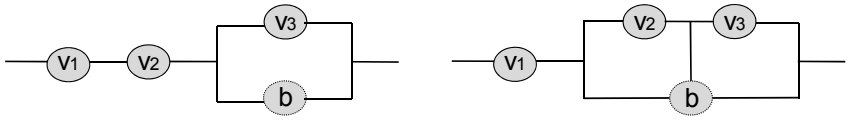
$$\begin{aligned} z_1 &\leq x_{v_1,n_1} \\ z_1 &\leq x_{v_2,n_1} \\ z_1 &\geq x_{v_1,n_1} + x_{v_2,n_1} - 1 \end{aligned} \tag{6.11}$$

The same applies to z_2, and to all other terms in Equation (6.10). Clearly this linearization process leads to an exponential increase in the number of variables and constraints, thereby nullifying any benefit incurred by it as the execution time of the model will increase dramatically. On the other hand, by relaxing the integrality of the node embedding and



(a) VNE (R = 99.88%) (b) Backup for V_3 (c) VNE with Protection Domains

Figure 6.4: Availability Analysis of Protection Domains



(a) $D = \{v_1, v_2, b\}$ (b) $D = \{v_1, v_2, b\}, \{v_1, b, v_3\}$

Figure 6.5: Protection Domains

link embedding variables ($x_{v,n}$ and $y^e_{i,j}$ respectively), the problem becomes in the form of a Geometric Programming (GP) [140] problem, which can be solved in polynomial time; however that will only provide us with a lower-bound. Given the NP-Hard nature of the availability-aware VNE problem, we propose JENA: a novel Tabu-based [103] search algorithm to provide a scalable approach for performing VNE with just-enough availability. The details of JENA are provided in Appendix 9.2.

6.5 Protection Domains for bare-bone VNs

6.5.1 Computing VN Availability with Protection Domains

Now, it could happen that no embedding solution can be found for the bare-bone VN that satisfies the service's availability demand. This occurs when the highly available servers are fully saturated; and the remaining feasible nodes[4] fail to satisfy the requested availability. For instance, consider the VN presented in Figure 2(b) embedded in the substrate network presented in Figure 4(a); here, observe that the highly available servers are fully exhausted, hence the mapping of the aforementioned VN on the remaining feasible nodes yields a 99.88% availability measure, which violates the requested availability of 99.95%. One way to overcome this limitation is by augmenting the VN with redundant nodes. The impact of

[4]Feasible physical machines refer to the active machines that have enough resources to satisfy the demands of the VMs.

augmenting a VN with redundant nodes (VMs) is additive on its overall availability. Since now, the VN's availability in no longer restricted to the availability of its primary VMs's hosts, but also to that of backup/redundant node(s)'s host(s) as well.

When a VN request is augmented with a redundant (backup) node, this redundant node is designated to protect a subset (or all) of the primary virtual nodes (VMs). This is achieved by provisioning the backup VM with enough computing resources to assume the breakdown of any VM in the set of protected VMs. Further, the backup node also requires bandwidth to resume communication with other VMs; hence backup virtual links must be provisioned between the backup node and the neighbors of each protected VM (with sufficient bandwidth resources). For instance, in the event where the bare-bone VN in Figure 2(b) is augmented with a backup node b to protect virtual node v_3 (as shown in Figure 4(b)), b will be provisioned with 2 units of computing resources, and will connect to v_1 (v_3's neighbor) via a virtual link with 2 units of bandwidth demand.

We denote this redundancy-enabled protection as a *protection-domain*. In the previous example where b is designated to protect v_3 (as shown in Figure 4(b)), this redesign will subsequently yield a protection domain $d = \{v_1,v_2,b\}$. Hence, the availability of this VN becomes:

$$\begin{aligned} A = Prob\{&each\ physical\ server\ hosting\ an\ unprotected\ VM\ \in\ VN\ is \\ &available,\ or\ any\ protection\ domain\ is\ available\} \\ &= a_{v_1}.a_{v_2}.a_{v_3} + a_{v_1}.a_{v_2}.a_b.(1 - a_{v_3}) \end{aligned}$$

Going back to our example, by mapping b on the remaining available machine F (as shown in Figure 4(c)), the overall availability of the VN becomes 99.89%. Note that the availability improvement provided by protection domain $d = (v_1,v_2,b)$ was not sufficient to achieve the availability requirement of 99.95%. In this regard, the cloud provider can decide to add more protection domains by protecting more primary VMs; for example, if b is set to protect v_2 and v_3 (as shown in Figure 5(b)), then b must now be provisioned with 3 units of computing resources (max computing demand of v_2 and v_3), and again b only needs to connect to v_1 (shared neighbor of v_2 and v_3) with 2 units of bandwidth resources. Hence, the set of protection domains becomes $D = [\{v_1,v_2,b\},\{v_1,b,v_3\}]$. Subsequently, the availability of this VN becomes

$$A = a_{v_1}.a_{v_2}.a_{v_3} + a_{v_1}.a_{v_2}.a_b.(1 - a_{v_3}) + a_{v_1}.a_b.a_{v_3}.(1 - a_{v_2})$$

By protecting both v_2 and v_3 the overall availability of the VN will reach 99.95%.
To generalize this, when a virtual node is augmented with k backup nodes protecting V' primary VMs in a given VN request, the availability of this VN can be quantified as the probability that all primary V virtual nodes are available, or any protection domain $d \in D$ is available, which can be expressed as follows:

$$A = \prod_{i \in V} a_i + \sum_{d \in D} \prod_{i \in d} a_i (1 - a_j) \quad \forall j \in V : \{j \neq i\} \tag{6.12}$$

Given an initial greedy embedding solution for a bare-bone VN whose availability is below A_{req}; the problem of provisioning protection domains while minimizing backup bandwidth ($\overline{BW}$) can be expressed as follows:

$$Min \quad A + \overline{BW}$$

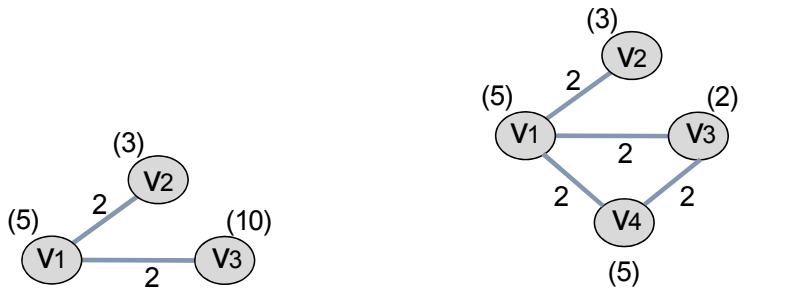
Subject to

$$A \geq A_{req} \tag{6.13}$$

$\overline{BW}$ represents the total amount of bandwidth provisioned for backup traffic; that is the bandwidth provisioned to connect the backup node to the neighbors of the primary VMs it protects. Hence, it is dependent on the placement of the added backup/protection nodes, as well as the protection assignment of each backup node to the various critical primary VMs. It should be noted that this model is generic, in that it adds as many protection domains as needed until it satisfies the required availability. Further, it can also be classified as an MINLP owing to Equation (6.12).

6.5.2 Protection Policies for Protection Domains Provisioning

Given that the problem of provisioning protection domains is complex in nature (as shown above), a more practical approach is to iteratively add a single protection domain until the requested availability is satisfied. Subsequently at every iteration, the only variable would be the placement of the backup node. However, the challenge becomes in deciding which critical VM this backup will protect. Hence, we propose two main *protection policies*: node residing on the most vulnerable host, or the node with the lowest "importance index". A node's importance index is denoted by the sum of the bandwidth demands on its adjacent links. The intuition behind these selection-policies are two-fold : while protecting the most vulnerable node achieves the highest overall availability improvement, the incurred



(a) Increase in Resource Demand (b) Arrival of New Network Component

Figure 6.6: Elastic VN Requests

bandwidth cost to connect the backup to its adjacent nodes may yield significant backup footprint, particularly if this node exhibits a high importance index. Conversely, protecting a node with a low importance index minimizes the incurred backup bandwidth cost, but may not yield a significant availability improvement, thereby requiring the addition of more protection domains. As will be shown through our numerical evaluation, selecting the best protection policy is highly correlated with the network state at the time of the embedding. Hence, we let the cloud provider dictate (alternate) the protection policy after assimilating the network's state via network monitoring tools (e.g. [81]).

6.6 Reliable Elastic Services

As previously mentioned, hosted services or network functions may scale up/down their requirements over time. We denote $\widetilde{G}^v = (\widetilde{V}, \widetilde{E})$ as a request to scale-up or down G^v at time t_1, with an availability requirement $\widetilde{A}_{req}$. A VN scaling request could enfold one or many of the following changes:

- Increase/Decrease in resource demands: This change typically involves the VMs in $\widetilde{V}$, such that $V \cap \widetilde{V} \neq \emptyset$, and/or the virtual links in $\widetilde{E}$, such that $E \cap \widetilde{E} \neq \emptyset$. For instance, Figure 6(a) illustrates the case where v_3's resource demand increases from 2 to 10 units.

- Arrival/Departure of New Network Component(s): This change typical involves VMs in $\widetilde{V}$, such that $V \cap \widetilde{V} = \emptyset$, and/or the virtual links in $\widetilde{E}$, such that $E \cap \widetilde{E} = \emptyset$. Figure 6(b) illustrates the case where an additional VM v_4 has been added to the original VN request in Figure 2(b), and needs to communicate with v_3 with 2 units of bandwidth.

- Service Class Upgrade/Downgrade: A_{req} at time t can change into $\widetilde{A}_{req}$ at time $t_1 > t$, such that $\widetilde{A}_{req} \geq A_{req}$ or $\widetilde{A}_{req} \leq A_{req}$.

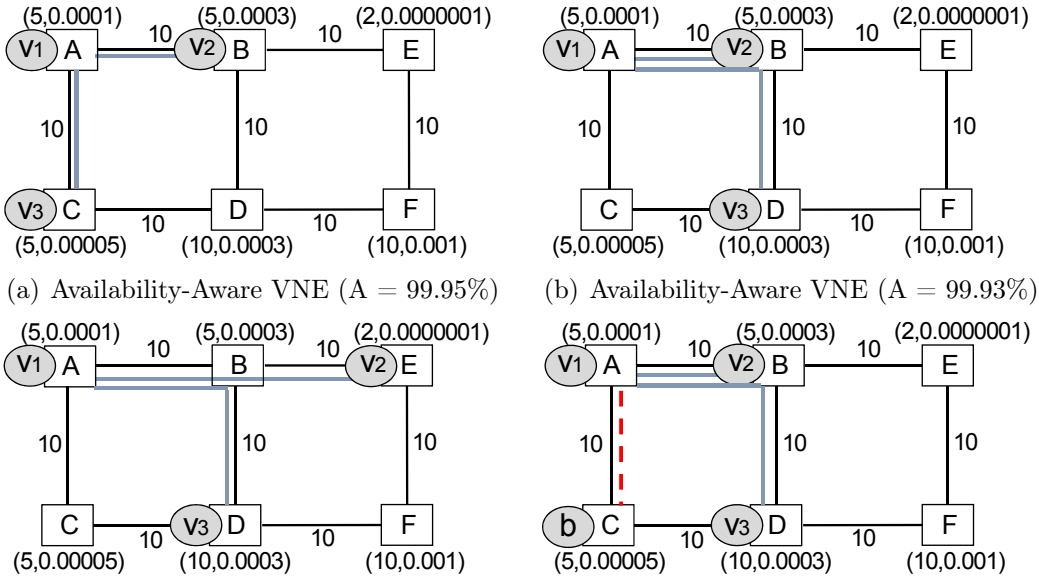
Problem Definition 6.2. *Given a VN request G^v = (V,E) with an availability requirement A_{req} at time t_0, which scales (up/down) into $\widetilde{G}^v$= ($\widetilde{V}$, $\widetilde{E}$) at time t_1 with an availability requirement $\widetilde{A}_{req}$; find the optimal reconfiguration of s into $\widetilde{s}$ such that the resource demands of $\widetilde{G}^v$ are met, while satisfying $\widetilde{A}_{req}$ and minimizing the overall reconfiguration cost.*

Here, reconfiguration cost reflects the amount of resources needed to host $\widetilde{G}^v$, as well as any service disruption/downtime that G^v might undergo in transition. In this work, we only consider the case of VMs resource demands increase, new network components arrival, and/or service class upgrade; and we handle VN scale-down requests by simply releasing the provisioned resources.
When a VN scales up into $\widetilde{G}^v$, it may happen that the initial reliable VNE solution is no longer feasible; e.g. a VM requires more resources beyond the residual capacity of its current host. In this regard, the cloud operator must reconfigure/update the initial embedding solution to satisfy the scale-up request. We denote V' as the set of VMs whose placement must be reconfigured; that is either existing VMs that demand additional resources beyond the residual capacity of the current hosts; or newly added VMs. Hence, the problem can now be seen as a quest for finding the lowest-cost placement of the VMs in V' that satisfies the availability demand of the VN in question.

Theorem 6.2. *The problem of managing reliable VN scaling requests is NP-Hard.*

Proof. We prove that the problem at hand is NP-Hard by restriction. Proof by restriction [139] consists of showing that the problem under consideration has at least one specific instance that is known to be NP-Hard. Indeed, the problem of reconfiguring a reliable VN scaling request is equivalent to solving the NP-Hard Generalized Quadratic Assignment Problem (GQAP) [141] when the availability requirement is 0. Since the latter has been proven to be NP-Hard [142], and represents an instance of the problem of managing reliable VN scaling requests; this leads us to conclude that the latter is also NP-Hard. □

When managing a scaling request, the cloud provider must make the necessary reconfigurations to accommodate the requested changes. This can be achieved by treating $\widetilde{G}^v$ as a newly arrived VN request, and subsequently it will be re-embedded and provisioned with resources from scratch. However, this will incur a high service disruption since even the unmodified VMs will be suspended and migrated. Alternatively, migration can be performed only for a subset of the VMs to meet the demands change while satisfying $\widetilde{A}_{req}$.



(a) Availability-Aware VNE (A = 99.95%)

(b) Availability-Aware VNE (A = 99.93%)

(c) Migration-Scheme (A = 99.96%)

(d) Redundancy-Scheme (A = 99.96%)

Figure 6.7: Benefits of Migration and Redundancy-Aware Approach

Indeed, when it comes to services with low-tolerance to disruption, this may be a more attractive solution for the cloud provider to avoid penalties for downtime experienced by this service during the re-embedding [121]. Moreover in some situations, migration alone might not be possible (services with no tolerance to disruption), insufficient (fails to reconcile the availability breach), or too costly (spreads the VMs too far apart, yielding longer substrate paths to route the traffic between them). Hence, in the following section we provide several illustrative examples to highlight the benefits of a migratory scheme to accommodate VN requests changes, as well as the importance of augmenting the VN with protection domains to circumvent cases where migration fails to reconcile the availability breach, or when the resultant post-migration solution is too costly.

6.6.1 Motivational Examples

6.6.1.1 Benefits of a Migration-Aware Approach

When managing a scaling request, the original embedding solution (resource allocation) may fail to meet the requested changes. For instance, consider the case where the bare-bone VN in Figure 2(b) is initially hosted in a given substrate network as shown in Figure 7(a), with a just-enough availability of 99.95%. At time t_1, the VN scales up by demanding additional resources for VM v_3, as shown in Figure 6(a). Here, we assume that the evolved

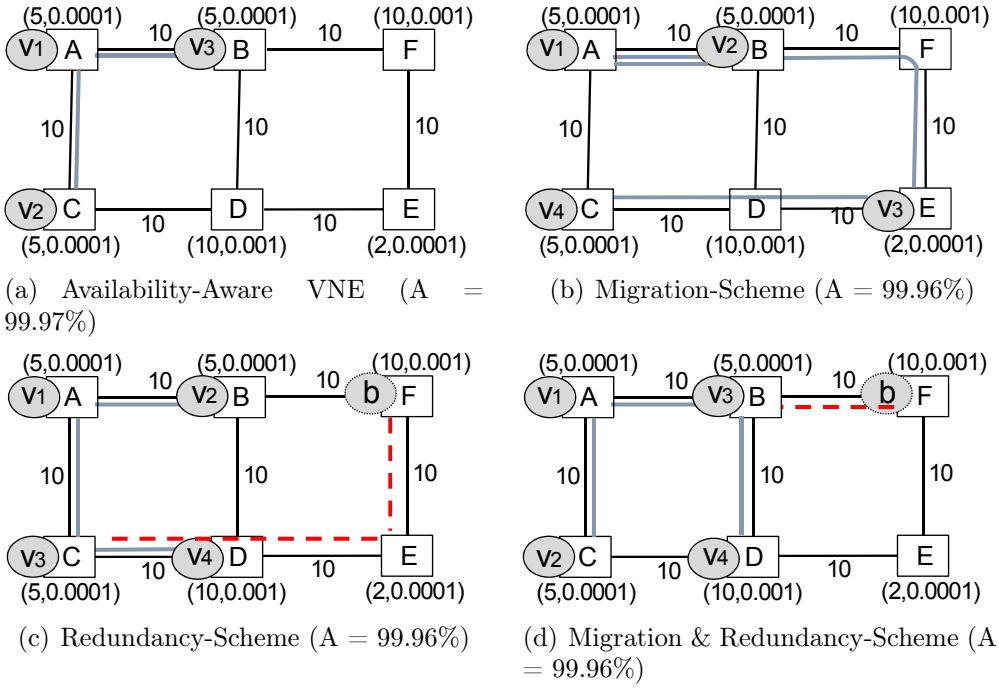
VN maintains its initial availability requirement of 99.95%($\widetilde{A}_{req} = A_{req}$). Hence at time t_1, the initial embedding solution presented in Figure 7(a) is no longer feasible since it fails to satisfy the requested resource demand increase. Here, the cloud provider must reconfigure the existing embedding solution to find a new host for v_3 that has enough resources to accommodate the new resource demand of 10 units. Observe in Figure 7(a) that v_3 can either migrate to substrate nodes D or F. Either way, an availability breach will occur since both these substrate nodes have a lower availability than v_3's current host. In fact, if substrate node D was chosen to host v_3 (as shown in Figure 7(b)), the overall availability of this VN will decrease to 99.93% $<$ 99.95%. Note that this availability breach will translate into 8 additional minutes of downtime per month above the negotiated threshold.
Hence, the cloud provider can either decide to reject this request, thereby losing revenue, or shift (migrate) some of the existing (unmodified) VMs to new hosts with higher availability in order to reconcile the availability breach. Indeed, migrating virtual node v_2 to substrate node E (as shown in Figure 7(c)) will improve the VN's availability to 99.96%, thereby alleviating the former infraction. However, this availability restoration comes at the expense of a service disruption while migrating v_2.

6.6.1.2 Benefits of a Redundancy-Aware Approach

When dealing with services which are highly intolerable to disruption, or in the event where migration is not feasible due to absence of available and/or more reliable hosts, then the migration-aware approach is no longer a valid option. Alternatively, protection domains can be added in the attempt to improve (reconcile) the availability. Going back to our example in Figure 7(b), by augmenting the VN in Figure 6(a) with a backup node b to protect VM v_2, a protection domain $d = \{v_1,b,v_3\}$ will be created. Hence by placing b on substrate node C (Figure 7(d)), the new overall availability of this enhanced VN (Equation (6.12)) becomes 99.96%. Note that this availability improvement incurs backup footprint to provision this redundant node, which remains idle until a failure occurs.

6.6.1.3 A Joint Migration and Redundancy-Aware Reconfiguration

Clearly, there exists a trade-off between a migration and a redundancy-enabled reconfiguration. Hence, the cloud provider must weigh-in the various options in order to find the most cost-efficient solution. In some situations, combining the benefits of both migration and redundancy can be explored to achieve the lowest overall bandwidth cost. To better illustrate this, consider again the bare-bone VN in Figure 2(b) embedded on the substrate network in



(a) Availability-Aware VNE (A = 99.97%)

(b) Migration-Scheme (A = 99.96%)

(c) Redundancy-Scheme (A = 99.96%)

(d) Migration & Redundancy-Scheme (A = 99.96%)

Figure 6.8: Joint Migration and Redundancy-Aware Reconfiguration

Figure 8(a) with $A = 99.97\%$. Further, consider that this VN scales-up by requesting an additional VM v_4 as shown in Figure 6(b). Here, the cloud provider needs to accommodate this VN change by finding the optimal embedding of the new node that yields the overall lowest reconfiguration cost while satisfying the requested availability $\widetilde{A}_{req} = 99.95\%$ (assume the availability does not change in this example). The most straightforward solution is to embed v_4 on either substrate node D or F, since they are the only nodes with enough resources to accommodate v_4. However, by doing so, the overall availability of this VN becomes 99.87%, thereby violating $\widetilde{A}_{req}$.

If the migration-aware approach is employed, VM v_3 can migrate to substrate node E (since E is the only host with a higher availability than any current host in the original embedding solution). Subsequently, v_4 can be placed on substrate node C (as shown in Figure 8(b)), improving the availability back to 99.96%, with an overall bandwidth cost of 12 units. On the other hand, the redundancy-enabled approach can add a backup node b to protect v_4 (the VM residing on the host with the weakest availability), as shown in Figure 8(c), also improving the availability back to 99.96%, and with an overall bandwidth cost of 12 units. Conversely, combining these two approaches by swapping the hosts of v_2 and v_3 (migrating v_3 to B and v_2 to C), and adding a redundant node b to protect v_4 (as shown in Figure 8(d)), the overall availability of this solution becomes 99.96% with an overall bandwidth cost of

just 8 units. That is a 33% lower cost than the migratory and redundancy-enabled schemes, at the expense of migrating two (unmodified) VMs and adding backup footprint to provision b.

6.7 ARES: Availability-Aware Reconfiguration Scheme

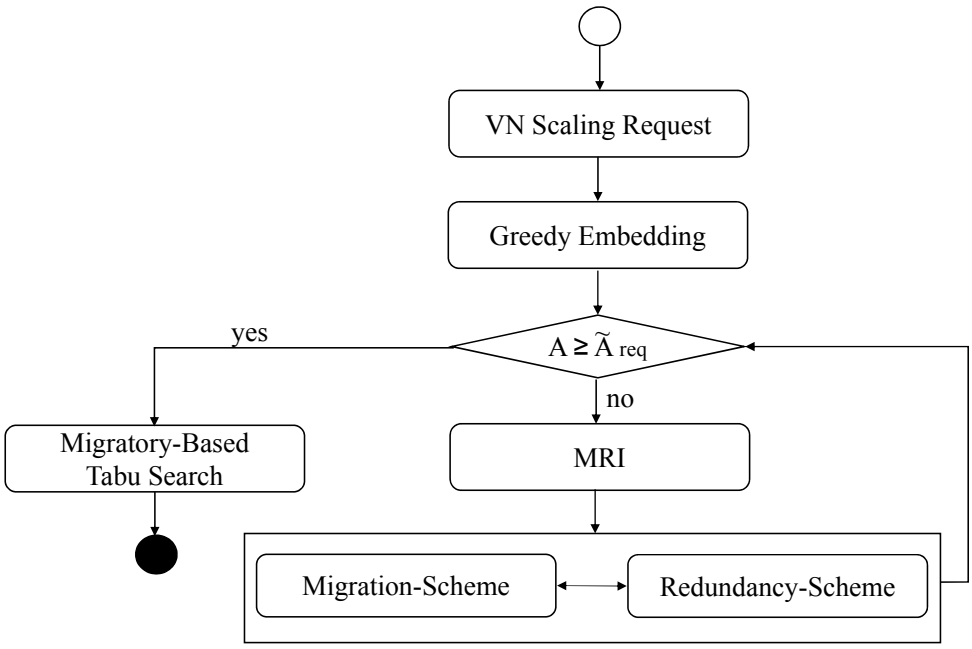


Figure 6.9: ARES Approach

Given the benefits and apparent trade-off between a migration- and a redundancy-enabled reconfiguration scheme, we propose ARES: our availability-aware reconfiguration scheme for managing VN scaling requests, which harnesses the benefits of both migration and protection domains adhesion. Our proposed approach, handles VMs resource demands increase and/or arrival of new VM instances, with or without service class upgrade. The procedural details of ARES are presented in Figure 6.9. It starts by receiving a request to scale up/down a VN $G^v = (V, E)$. We denote V' as the amended set of virtual nodes at time t_1; that is the set of virtual nodes with increased resource demands that cannot be satisfied by their current host, and the set of newly arrived instances. Let $\hat{V} \subset V$ denote the set of unmodified VMs in the original VN request. Subsequently, the cloud provider must decide the optimal placement for V', while satisfying the required availability $\widetilde{A}_{req}$. It starts off by employing an initial greedy embedding for V' by scouting the substrate network for the set of substrate nodes that have enough resources to host each $v' \in V'$, and then selecting the ones with the

highest availability (while avoiding conflicting placement with $\hat{V}$). If no substrate nodes with enough residual capacity can be found to host any $v' \in V'$, then the algorithm terminates for resources scarcity. Conversely, the resultant embedding solution s_N will be returned.
In the event where the obtained greedy embedding solution satisfies $\widetilde{A}_{req}$ with a much larger availability than requested (availability over-provisioning), the embedding solution must be adjusted by shifting some VMs to less reliable hosts such that "just-enough" availability is achieved. Hence, we run our proposed Tabu-based search again (as presented in Section 9.2) to enhance on the greedy-embedding solution, and find a lower-cost mapping which provides "just-enough" availability with minimal service disruption. However, we slightly modify our initial Tabu-based search (Appendix 9.2) to enable migration of unmodified VMs in $\hat{V}$, only if this migration yields a lower embedding cost. Note that since migration yields service disruption, we modify the cost function (Equation 9.5) to include a migration penalty. Hence, Tabu will try to enhance on the greedy embedding solution without incurring a large service disruption.
On the other hand, if the greedy embedding solution of V' fails to satisfy $\widetilde{A}_{req}$, then the cloud provider must decide to either migrate a subset of $\hat{V}$, or introduce redundant nodes to improve the VN's availability. Clearly we need to decide which node to either migrate or protect. Given the greedy embedding solution with an availability breach, the choice to migrate or protect one node over the other severely impacts the incurred cost, as well as the achievable availability improvement (as presented in Section 6.6.1). Thus, we delegate this choice to our Migration/Redundancy Iterator (MRI) algorithm. The role of the MRI is to determine the best course of action based on the substrate network's state at the time when the scaling request is received, as well as the SLA of the associated VN. The MRI is invoked iteratively while $A < \widetilde{A}_{req}$. Once an embedding solution that exceeds the requested availability is found, then the migratory-aware variation of Tabu is executed, to enhance on the solution. The procedural details of the MRI are elucidated below.

6.7.1 Migration/Redundancy Iterator (MRI)

In the event where the greedy embedding solution fails to satisfy $\widetilde{A}_{req}$, the MRI is invoked to decide whether to migrate one of the VMs in $\hat{V}$, or augment the VN with a redundant/backup node to improve the VN's availability. This decision depends on many factors, mainly the state of the substrate network at the time when the reconfiguration is performed, as well as the VN's tolerance to service disruption. For instance, if the VN is highly intolerable to service disruption, then a VM migration will incur grave penalties. Alternatively, if the

substrate network has scarce resources, then investing in redundant (idle) resources might be an injudicious/infeasible move at that moment. Hence, we set two weight values α_m and α_r for migration and redesign, respectively. Before invoking the MRI, the cloud provider can adapt the value of these weights after assimilating the network's state and the VN's SLA. Subsequently, if α_m is greater than a predefined threshold, MRI will invoke the redesign-aware scheme, otherwise the migration-aware scheme will be invoked. If the weight on α_m and α_r is comparable, then the MRI will pick the move that yields the lowest overall embedding cost in terms of bandwidth. Here, the challenge becomes in deciding which critical VM to protect or migrate at every iteration. Hence, we adopt the same policies proposed in Section 6.5.2. Let $\hat{v}$ denote the VM selected based on the dictated protection policy, hence the problem of finding the optimal reconfiguration can be formulated as follows:

$$Min \ -\gamma(A - A_0) + \ (1-\gamma) \sum_{(i,j)\in L} (t_{i,j} + \hat{t}_{i,j}) \tag{6.14}$$

Subject to

$$\sum_{n\in N} z_n \leq 1 \tag{6.15}$$

$$z_n + \sum_{v\in V} x^0_{v,n} \leq 1 \quad \forall n \in N \tag{6.16}$$

$$\sum_{i:(i,j)\in L} \hat{y}^v_{i,j} - \sum_{j:(j,i)\in L} \hat{y}^v_{(j,i)} = z_n - x^0_{v,i} \quad \forall i \in N, \ v \in Adj(\hat{v}) \tag{6.17}$$

$$\hat{t}_{i,j} = \sum_{v\in V} \hat{y}^v_{i,j}.b'_{v,\hat{v}} \quad \forall (i,j) \in L \tag{6.18}$$

$$\sum_{n'\in N:\{n'!=n\}} m_{n,n'} \leq 1 \quad \forall n \in N : \{x^0_{\hat{v},n} = 1\} \tag{6.19}$$

$$\sum_{i:(i,j)\in L} y^{e:(v,\hat{v})}_{i,j} - \sum_{j:(j,i)\in L} y^{e:(v,\hat{v})}_{j,i} = x^1_{\hat{v},i} - x^0_{v,i} \ w_{\hat{v}} \quad \forall i \in N, \ v \in Adj(\hat{v}) \tag{6.20}$$

$$w_{\hat{v}} \geq \sum_{n\in N:\{x^0_{\hat{v},n}=1\}} \sum_{n'\in N:\{n'\neq n\}} m_{n,n'} \tag{6.21}$$

$$\sum_{n'\in N'} (z_{n'} + m_{n,n'}) = 1 \quad \forall n \in N : \{x^0_{\hat{v},n} = 1\} \tag{6.22}$$

The presented model has a weighted objective, striking a balance between migration and redesign. The objective function (6.14) aims to find the solution that yields the highest availability improvement with the lowest overall embedding cost; where A_0 (given) and A

(computed via Equation 6.12) denote the availability of the pre- and post-reconfiguration solution, respectively, and $t_{i,j}$ and $\hat{t}_{i,j}$ denote the primary and backup traffic routed on each link $(i,j) \in L$. γ is used to adjust the weight between these two objectives. Constraints (6.15)-(6.18) indicate the mapping of the backup node and backup links. Let $x^0_{v,n}$ denote the initial embedding solution. Constraint (6.15) indicates the placement of the backup node ($z_n \in \{0,1\}$), while Constraint (6.16) ensures non-conflicting placement between primary and backup nodes. Constraint (6.17) performs the flow conservation constraint for the backup links, whereas Constraint (6.18) measures the backup bandwidth incurred on each substrate link. Constraints (6.19) decides the new host for $\hat{v}$ (in case of migration, $m_{n,n'} \in \{0,1\}$), and Constraint (6.20) re-routes the primary virtual links of $\hat{v}$ accordingly ($y^e_{i,j} \in \{0,1\}$). Constraint (6.21) indicates whether the selected node has migrated or not ($w_{\hat{v}} \in \{0,1\}$), and Constraint (6.22) forces the model to make a single move, by either migrating $\hat{v}$, or placing a backup node to protect it. Note that additional constraints must be placed to avoid conflicting node placement and substrate network capacity violation.

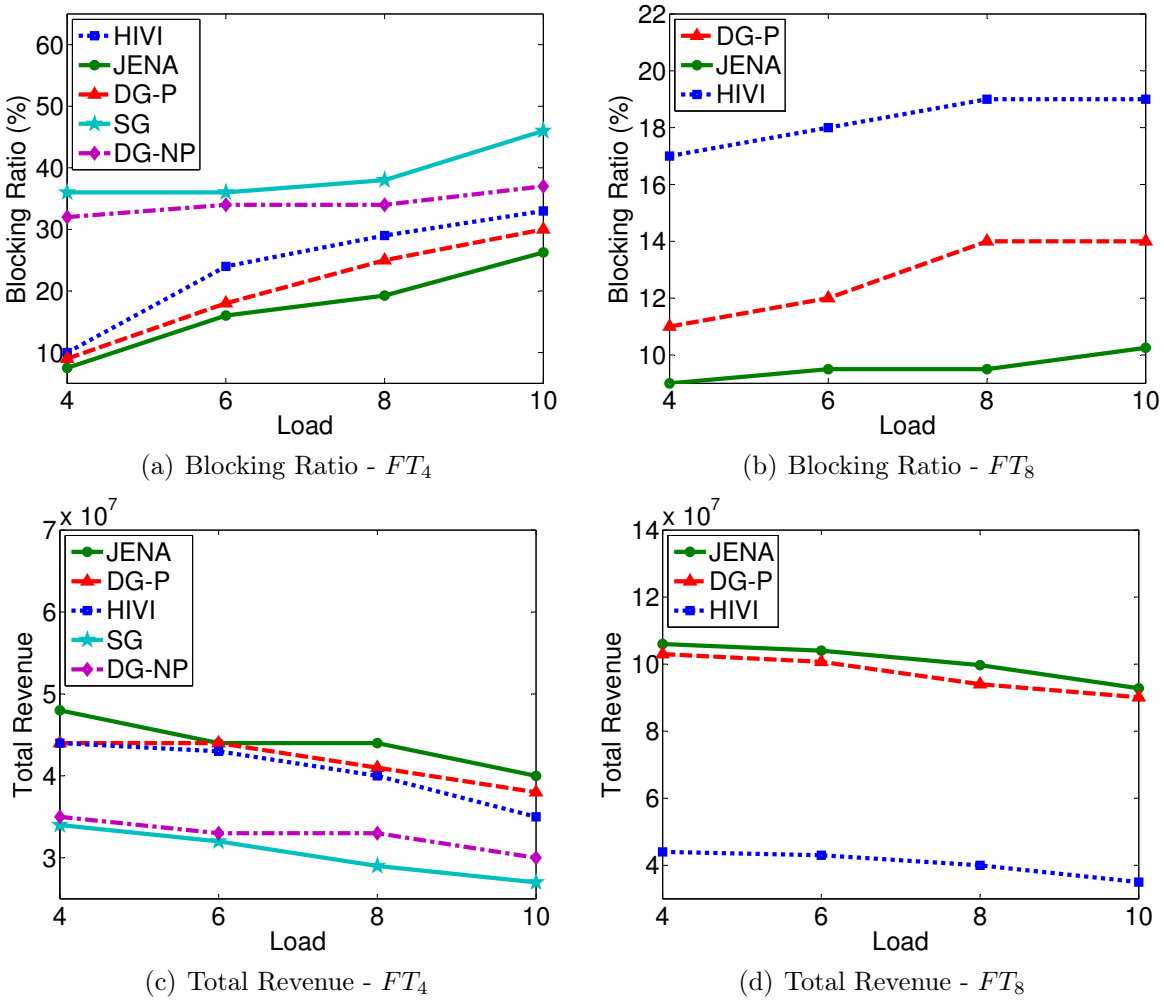
6.8 Numerical Results

In this section, we numerically evaluate the performance of RELIEF, first by looking at how the availability-aware embedding module performs compared to peer and benchmark algorithms, and second by analyzing the reconfiguration module as the embedded VNs scale over time. Hence, we separate these two evaluation concerns. We vary the availability of the substrate nodes between [0.9999,0.999999], and we vary the availability requirement of the VNs between [0.999,0.99999]. Throughout our numerical results, we let the size of the VNs vary between [2-20] VMs.

6.8.1 Comparative Analysis of JENA Module

To evaluate the performance of JENA, we look at three main metrics: blocking ratio, revenue and execution time. Blocking ratio refers to the percentage of VNs rejected out of the total number of VN embedding requests; and revenue is computed as follows: $\sum_{r \in R} \sum_{v \in V} c'^r_v \ \rho_r$; where ρ_r denotes the price per unit of leased resource r for a period of time. We adopt the FatTree (k=4) and (k=8) [28] networks (denoted as FT_4 and FT_8, respectively), since FatTree is a commonly adopted topology for data centers. We begin by comparing it against HIVI [59]: a peer embedding technique extracted from the literature, as well as three benchmark algorithms: namely, a static greedy (denoted as SG), and a dynamic greedy with and

without protection (denoted as DG-P and DG-NP, respectively). Here, SG consists of treating each incoming mapping request in a greedy fashion, that is sorting the set of substrate nodes in descending order of their availability measure, and then attempt to embed the VMs one-by-one. The DG-NP differs from the former in the fact that it aims to provide just-enough availability. Hence, upon performing a greedy embedding for an incoming VN, the DG-NP attempts to remove any over-provisioning by iteratively downgrading the VMs from highly available servers to ones with lower availability, while satisfying the requested availability guarantee. Finally, DG-P performs the same embedding routing as DG-NP, and in addition is equipped with the ability to add protection domains in case no feasible solution can be found for a bare-bone VN.



(a) Blocking Ratio - FT_4

(b) Blocking Ratio - FT_8

(c) Total Revenue - FT_4

(d) Total Revenue - FT_8

Figure 6.10: Comparative Analysis of JENA

1. **Execution Time:** First, we look at the execution time achieved by each one of the aforementioned techniques, the results are shown in Table 6.1. We randomly generate

Table 6.1: JENA Runtime Analysis(ms)

	JENA-V	JENA-I	DG-P	HIVI	SG	DG-NP
FT_4	16	42	5	2	1	2
FT_8	43	78	35	14	12	21

a set of 100 VNs, and we set the load to 10. Clearly, we observe that all of the aforementioned techniques execute in the order of milliseconds. Here, we also compare the execution time of our embedding technique under different protection policies: vulnerable (denoted as $JENA-V$), and importance index (denoted as $JENA-I$). Observe how the importance index policy exhibits a higher execution time (61% higher over FT_4) than the vulnerable host policy, since the former requires more iterations to enhance on the availability measure than the latter.

2. **Blocking Ratio:** Next, we look at the blocking ratio; since a high execution time does not necessarily indicate a good performance in terms of network resource utilization and admissibility. The results are illustrated in Figure 10(a) and 10(b). Clearly, we observe that JENA achieves the lowest blocking ratio among its peers over both FT_4 and FT_8. Indeed, for a load of 10 over FT_4, JENA achieves 23% lower blocking than DG-P, and at least 33% lower blocking than HIVI, SG and SG-NP.
   Further, we observe that over the FT_4 network, DG-NP achieves a lower blocking than SG (20% lower for a load of 10), which vows for the positive correlation between achieving just-enough availability and its impact on network resource utilization and network admissibility. Moreover, the leverage in alleviating availability violations via protection domains is highlighted by the higher admissibility achieved by each one of the redundancy-enabled techniques: HIVI, DG-P, and JENA. Finally, we can conclude that the systematic Tabu-based search that JENA adopts for finding the lowest-cost embedding solution with "just-enough" availability, allows it outperform both HIVI and DG-P.

3. **Total Revenue:** Finally, we look at the total revenue achieved by each of the aforementioned approaches (as shown in Figure 10(c) and 10(d)). Here again, we observe that our proposed embedding module achieves the highest total revenue as we vary the load over both substrate networks. Indeed, for FT_4 and a load of 4, JENA achieves 10% higher revenue than DG-P and HIVI, as well as 30% and 51% higher revenue than SG and DG-NP, respectively. Similar results can be observed over FT_8.

6.8.2 Comparative Analysis of ARES

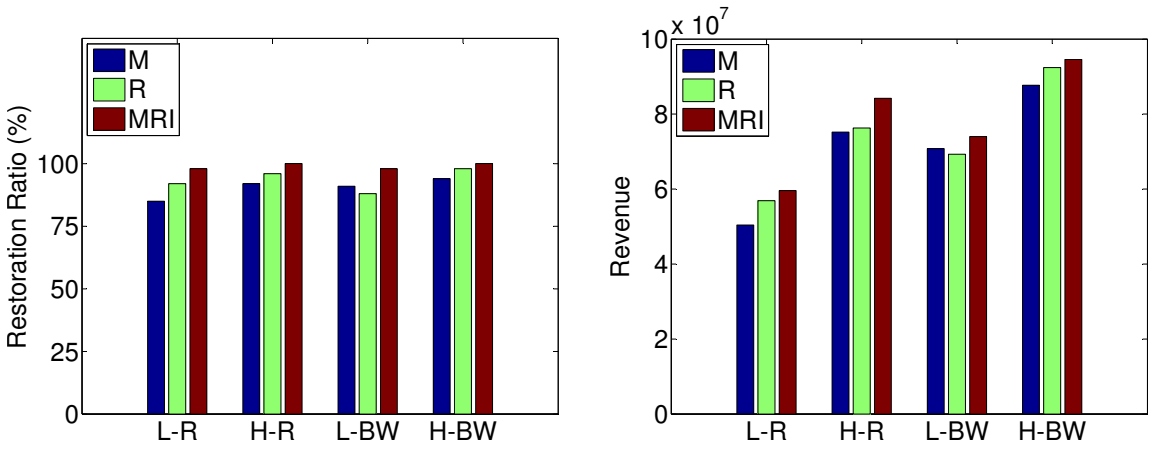
In this section, we numerically evaluate the performance of ARES against benchmark scaling techniques that are either migration or redundancy oblivious. We refer to these latter as "Iterative Migration" (denoted as M) and "Iterative Redesign" (denoted as R), respectively. Also, we distinguish between two selection-policies, that is either selecting the most vulnerable node or the node with the lowest importance index to migrate or protect. Here we adopt three main network topologies: FT_4, FT_8, and a randomly generated [118] substrate network with 60 nodes and 90 links (that we denote as RN), and we look at four main metrics: execution time, average cost, revenue, and admission. Further, to simulate scaling requests, we setup a random scaling trigger with a frequency metric F, where F defines the interval of random scaling requests that will occur over time. At every scaling trigger, we let P% of the VNs hosted to scale up/down. This scaling can be: an increase/decrease of resource demands, arrival/departure of VMs, service upgrade/downgrade, or any combination of them. Throughout our numerical evaluation, we let γ=0.5, and we set F, P, and the load to 30, 40%, and 8, respectively.

1. **Execution Time:** First, we begin by evaluating the execution time of the various reconfiguration schemes (as shown in Table 6.2). We generate a random set of 100 VNs, and we measure the average time to reconfigure the mapping solution of the various scaling requests. Here, we observe that all of these reconfiguration schemes execute in the order of seconds. Further, we observe that the selection-policy greatly impacts the execution time, where selecting the most vulnerable node to migrate/protect can reconciliate the availability breach in fewer iterations. For instance, we observe that for MRI over FT_8 with the importance index as the selection-policy, is 50% slower than the case where the vulnerable node policy is selected.

Table 6.2: ARES Runtime Analysis (ms)

	M		R		MRI	
	I	V	I	V	I	V
FT_4	0.075	0.068	0.172	0.104	0.178	0.112
FT_8	1608	546	2493	834	1850	938
RN	0.445	0.217	0.417	0.281	0.908	0.398

2. **Impact of Substrate Nodes Availability:** Next, we look at the impact of substrate



(a) Restoration Ratio (b) Total Revenue

Figure 6.11: Comparative Analysis of ARES - Impact of Substrate Nodes Availability and Network Bandwidth

nodes availability on the achievable restoration ratio by each of the aforementioned reconfiguration techniques. Restoration ratio denotes the percentage of successful reconfigurations out of the total number of scaling requests received. We adopt the random network RN, and we vary the range of the substrate nodes's availability within either L-R=[0.999,0.99995] or H-R=[0.9999,0.9999999], where L-R and H-R denote the low and high servers availability range, respectively. Here, we observe that the iterative migration M's restoration ratio drops from 95% to 87%, whereas R and MRI maintain an availability above 92% (as shown in Figure 11(a)). Hence, we can conclude that the effectiveness of the iterative migration scheme is highly dependent on the availability of the servers in the substrate network. To better reflect the impact of this restoration ratio drop, we look at the total revenue achieved by each of the proposed schemes, illustrated in Figure 11(b). Indeed, we observe that M achieves 11% lower total revenue than R, and 13% lower revenue than MRI.

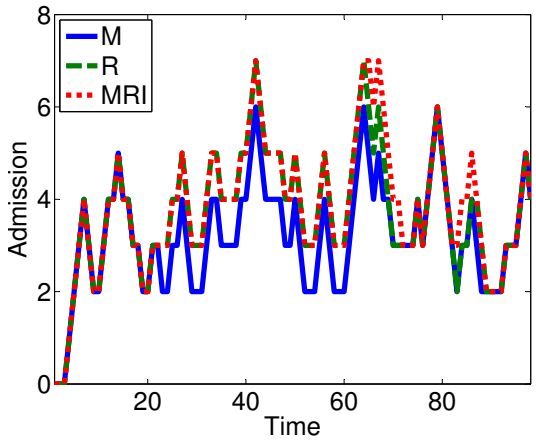
3. **Impact of Bandwidth Capacity:** Second, we look at the impact of substrate links capacity (as shown in Figure 11(a)), as we vary their size between 1 Gbps (denoted as L-BW) and 10 Gbps (denoted as H-BW). We observe that the restoration ratio of the iterative redesign scheme is negatively affected under the L-BW network state, with a restoration ratio drop from 98% to 88%, leading to a lower total revenue than that achieved by M and MRI (as shown in Figure 11(b)). On the other hand, under the H-BW network state, R outperforms M, while MRI achieves the highest gain under either one of these network states.

These results support our claim on the correlation between network's state and the selection policy undertaken by the cloud provider. Indeed, when the network is in a state of high traffic load (case of low bandwidth $L\text{-}BW$), it is more acute to avoid adding redundant nodes to avoid the fallout cost of backup footprint. On the other hand, when the network has a low traffic load (case of high bandwidth $H\text{-}BW$), then iterative redesign achieves better results than iterative migration, since the latter is restricted to the availability of the primary hosts. Clearly combining the benefit of migration and redesign, using our proposed MRI, enables to circumvent the limitations of network traffic load and primary hosts availability, as observed by the gain achieved in terms of restoration ratio and revenue under different network states.

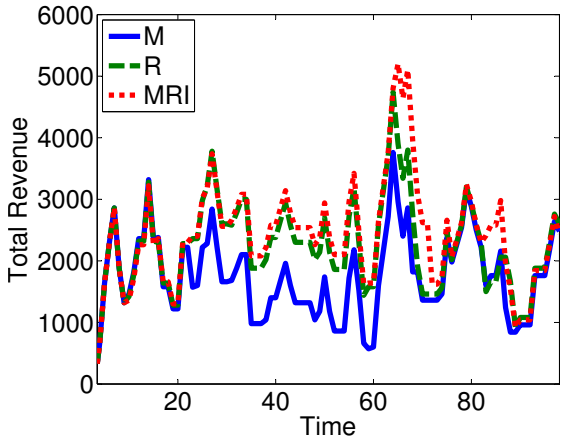
4. **Admission, Cost and Revenue:** In this section, we look at the admission rate, total revenue, and average cost over time, for two substrate networks RN and FT_4 (as shown in Figure 12(a)-12(f)). Overall, we observe that MRI achieves the highest admission and revenue over time for both substrate networks. For instance in Figure 12(a), in the time slot between [40-60], we observe that though MRI and R exhibit the same admission rate, yet the average reconfiguration cost achieved by MRI is 7-20% lower than that achieved by R (as shown in Figure 12(c)). Indeed, this cost-efficient reconfiguration scheme enhances the network's admissibility, rendering a 14-25% higher admission over R in the time slot between [60-75], and an overall revenue gain that is 30-60% higher than that achieved by M, and 4-44% higher than that achieved by R over time. Similarly for the random network, MRI achieves a 20-77% higher revenue than M, and 2-69% higher revenue than R over time. Indeed, we can conclude that by harnessing the benefits of both migration and redundancy, the admissibility of the substrate network is enhanced, and subsequently the network operator's long-term revenue.

6.9 Conclusion

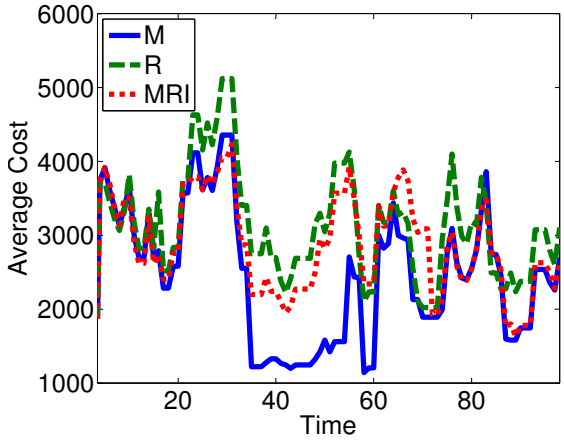
In this chapter, we proposed a novel reliable embedding and reconfiguration framework for elastic services in failure-prone data center networks. We proved that as opposed to existing work, our embedding module promotes better resource utilization as it avoids availability over-provisioning. Further, we showed that as VNs scale, their initial embedding solutions may fail to meet the requested changes, or even yield an availability breach. Hence, we mathematically formulated the problem, and provided several motivational examples to present the



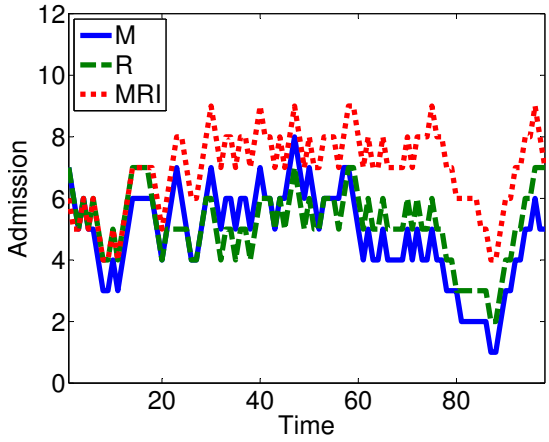
(a) FT-Admission



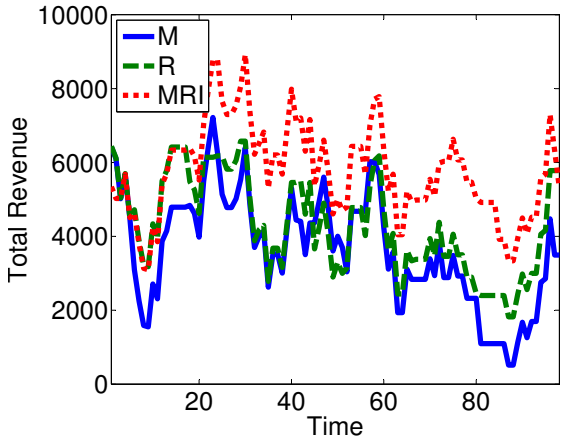
(b) FT-Revenue



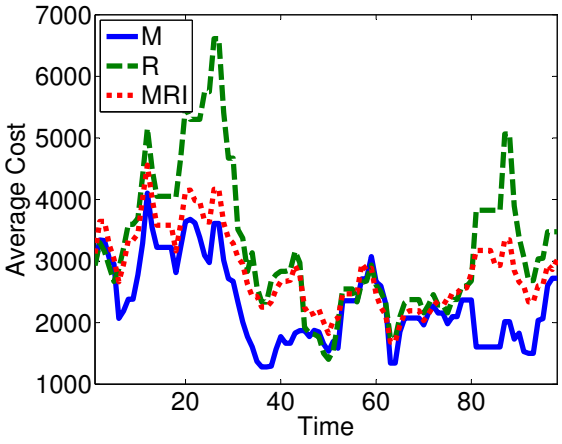
(c) FT-Cost



(d) RN-Admission



(e) RN-Revenue



(f) RN-Cost

Figure 6.12: Comparative Analysis of ARES - Admission, Cost, and Revenue

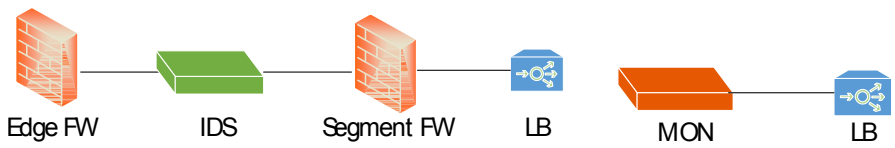
pros and cons of a migratory versus a redundancy-enabled scheme to reconfigure scaled VNs. Given the intricate interplay of these two schemes, we proposed ARES: an availability-aware reconfiguration module for elastic services that leverages the benefits of the redundancy and migratory schemes to achieve a low-cost reconfiguration with minimal service disruption. Our numerical results prove that our suggested approach achieves encouraging gains over migration or redundancy oblivious schemes.

Chapter 7

A Cut-and-Solve Based Approach for the VNF Assignment Problem

Middleboxes have recently become a ubiquitous element in operator's networks [143–147], with an abundance commensurate to the numbers of routers/switches in the network [148]. This "middlebox-outburst" is mainly due to the significant value-added services these latter provide to traffic flows, in terms of enhanced performance and security. Firewalls, Load Balancers, and Intrusion Detection Systems (IDSs) are some examples of middleboxes, also known as network functions (NFs), residing in today's networks. Each of these middleboxes, provide a function to serve the flows on their path from source (ingress) to destination (egress). E.g., firewalls filter traffic based on predefined rules, load balancers distribute the traffic to multiple destination hosts, IDS collect data for security checks, etc. Flows usually need to traverse multiple middleboxes in a predefined order; For instance, a packet must traverse an IDS before going through a Wide Area Network (WAN) optimizer [149] for encryption. This type of traffic flow traversal is referred to as *Service Function Chaining* (SFC), commonly prescribed as a traffic flow policy [150]. The IETF presents several SFCs use cases in data centers [151], mobile [152], and broadband [153] networks.

Typically, Middleboxes run on specialized hardware, which make them highly obstinate.



(a) Policy Chain 1 (b) Policy Chain 2

Figure 7.1: Service Function Chaining

Their location/placement in the network is bound to lose its efficiency over time, particularly as the traffic flow in the network is unpredictable; not to mention network bandwidth overconsumption when routing traffic from ingress to egress points that are too far from the middlebox location. Moreover, hardware NFs are expensive and vendor-specific, and network operators tend to over-provision in the number and type of hardware NFs procured to handle traffic surges [154]. This in turn yields significant overheads in terms of Cap-ex and Op-ex, to power, maintain, and configure them. Furthermore, given the short hardware life cycle and the fast-pace technological advancements, adding new services and functionalities or replacing existing hardware requires the procurement of more hardware NFs, which is a tedious and cumbersome procedure [61, 154].

Network Function Virtualization (NFV) is a promising new technology that aims at tackling the aforementioned limitations of hardware NFs [61]. It consists of decoupling the software from the hardware, yielding softwarized-middleboxes that can run atop any hardware commodity (standard servers, or routers/switches [155]). These Virtualized Network Functions (VNFs) can be instantiated on demand, and migrated anywhere in the network; thereby reducing the Cap-ex associated with middlebox over-provisioning. Moreover, these software-based NFs support multi-tenancy, and can be easily upgraded/modified [61]; thereby reducing Op-ex required to maintain/configure them, and greatly enhancing the time-to-market. Yet, NFV is still in its infancy, and there exists multiple challenges, reported by the ETSI group [61] and IETF [150], that must be addressed to facilitate and enhance its adoption.

Recently, NFV has received significant attention from the literature in response to these calls for actions. While some [156–161] targeted the VNF orchestration and management concerns, others aimed at facilitating the realization of NFV by developing virtualized software-based NFs platform e.g. ClickOS [154], NetVM [162], and routing function virtualization [163]. Further, to achieve the economy of scale promised by NFV, the placement of VNFs and the flow-to-VNF assignment emerged as equally important problems [61]. Indeed, suboptimal placement of VNFs may yield to inefficient resource utilization, as well as significant bandwidth overhead to route the traffic flow through them, therefore lowering the admissibility of the network. We denote the latter problem as the Virtual Network Function Assignment problem, which has been proven to be NP-Hard [147]. It consists of finding the optimal placement for the VNFs and the flow-to-VNF assignment to maximize the admission of policy-aware traffic requests. Given the NP-Hard nature of this problem, a large body of work emerged lately to tackle it. While some relaxed the problem by considering a single VNF placement [164–167], or a single policy class [168], others considered that the

traffic flows are known-apriori, and proposed Integer Linear Programming (ILP) formulation [149, 168–172] which runs in the order of hours. Moreover, a handful of heuristic-based methods [147, 173–176] emerged, with no guarantees on the quality of the obtained solutions. This chapter is concerned with the VNF assignment problem, and proposes to jointly address the problem of VNF placement and policy-aware traffic steering to maximize the number of flows routed across the network. Unlike previous work, and to keep track of the problem, we propose a cut-and-solve based approach. Hence, we decompose the problem into two subproblems: a master and a subproblem. The master is in charge of performing the VNF placement and assigning appropriate VNF instances to every flow; while the subproblem performs the policy-aware routing of every flow along its designated VNF instances. At every iteration, constructive piercing cuts are generated and added to the master to tighten its search space. We compared our proposed method against the ILP, as well as a heuristic-based method, and we show that our approach achieves optimal solution 700 times faster than the ILP-based formulation.

7.1 Related Work

Recently, a handful of contributions [146, 177, 178] have tackled the policy-aware traffic steering problem via hardware middleboxes. However, given the limitations of hardware middleboxes, attention converged towards enabling the support of network function virtualization. While some focused on providing an architecture for building and supporting softwarized-middleboxes, such as XOMB [179], ClickOS [154], and NetVM [162]; others proposed solutions to deploy and orchestrate VNFs [158–160].

An equally challenging problem that emerged along with NFV is solving the VNF assignment problem; that is the policy-aware traffic steering via virtualized network functions. Here, the location of the VNFs is undefined, and it is up to the network operator to decide on the number of VNF instances to deploy, such that the forwarding policies associated with incoming traffic requests can be fulfilled. PACE [176] is among the earliest work that addressed this problem; and given its NP-Hard nature, the authors tackled the placement of VNFs and the traffic steering of flows disjointly. Such an approach is bound to be costly, since the VNF placement greatly affects the traffic-steering and may very well yield to significant bandwidth consumption; particularly when the VNF instances are placed too far from the flows's ingress and egress nodes. The work in [166, 167] also considered the VNF assignment problem; however, the authors considered that every flow can be routed via a single network

function; further in [167] the authors relax the problem by assuming a tree-based network topology with a single path between every pair of substrate nodes. Other work [164, 165] also relax the problem by either considering unlimited network resources [164], or unlimited number of VNF instances [165].

Further, a handful of contributions consisted of ILP-based formulations [149,168–172], which are known to be fairly unscalable and computationally intractable. In [149], a Mixed Integer Quadratically Constrained Program (MIQCP) is proposed for finding the best placements of chained VNFs; while the work in [168] model the problem as an ILP, and considers a single policy class for all incoming traffic requests. In [169] and [171], the authors also formulate the problem as an ILP model. In [171], the authors propose a heuristic-based approach to guide the ILP solver towards near-optimal solutions, which runs in the order of minutes, and in [172] a sequential fixing-based heuristic is proposed to solve larger instances of the problem. In [170] the authors tackle the hybrid VNF assignment problem, where network functions can be either hardware or software; and propose an ILP model to solve it. Other work in the literature consisted of heuristic and meta-heuristic based approaches [147, 173–175], with no guarantee on the quality of the obtained solution. For instance, Lukovszki and Shmid [175] present a deterministic algorithm for solving the online VNF embedding problem, while Bari et al. [147] model the problem as a multi-stage graph, and sequentially solve the VNF assignment problem for every incoming flow request using a Viterbi-based algorithm. In [173], the authors proposed a simulated annealing based algorithm. Finally, in [174] the authors proposed an ILP model to solve the VNF assignment problem, as well as different heuristic variations for solving the problem for larger instances. One proposed heuristic (Heuristic-A[1]) consists of solving the VNF assignment problem disjointly, by sequentially finding the shortest path from the ingress to the egress node of every flow, then attempts to place the VNF instances required in the flow's policy along this path. An other proposed heuristic (Heuristic-B) consists of dividing the flow requests into groups, then iteratively solving the VNF assignment problem for each group using the ILP model. The remaining heuristics are a variation of Heuristic-B, and mainly differ in the way the flows are partitioned into different groups.

Our work is different since it provides an exact solution to the VNF assignment problem within considerably much faster runtime than the ILP. As opposed to heuristic-based work, our algorithm provides optimal solutions, and as shown in our numerical results is much more scalable than ILP-based formulations. Further our approach is topology-independent,

[1]A variation of this heuristic is used in our numerical analysis.

supports chains of multiple VNFs, and aims to solve the problem while accounting for the finite resource capacity of the substrate network, as well as the limited number of VNF instances allowed.

7.2 The VNF Assignment Problem

7.2.1 Network Model Overview

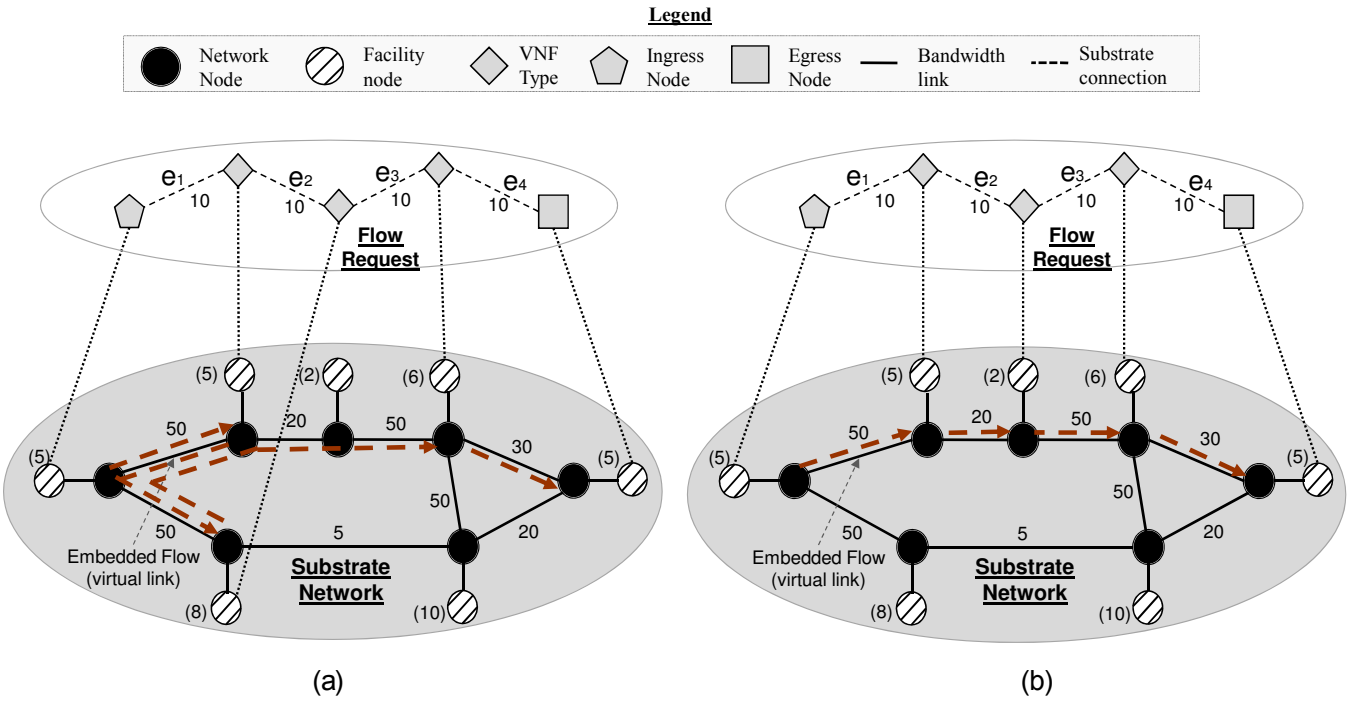


Figure 7.2: Network Model

In this section, we present a formal definition of the VNF assignment problem by describing the various components involved:

1. **The Substrate Network :**

   We represent the substrate network as an undirected graph, denoted by $G^s = (N,L)$, where N is the set of substrate nodes, and L is the set of substrate links. Each substrate node $n \in N$ is associated with a finite capacity[2], denoted by c_n . Similarly, each substrate link $l \in L$ has a finite bandwidth capacity, denoted by b_l.

[2]For the sake of simplicity, we only consider a single resource type (e.g., memory, CPU); however our work can be easily extended to account for multiple resource types.

2. **Virtualized Network Functions (VNFs):**
   VNFs represent the various network functions offered by the network provider. Let M denote the set of all VNF types (e.g., a firewall represent one VNF type). To deploy an instance of a VNF type m on a substrate node n, the latter must have enough resources to accommodate the former's resource requirements (e.g., memory and CPU needed to run an instance of a firewall). Hence, each VNF type $m \in M$ is associated with resource requirement w_m. Note that there can be multiple instances of a single VNF type, and different middlebox instances can be collocated on the same substrate node. Further, every deployed instance of a VNF type is associated with a processing capacity (p_m), which reflects the maximum traffic load each instance can accommodate. Here, the maximum number of instances allowed for any middlebox type is limited, e.g., by the number of licenses owned by the network provider [149], that we denote as K_m. Throughout this work, we consider that all VNF types are stateless [173](i.e., can be shared by multiple tenants), but our work can be easily modified to account for stateful VNFs.

3. **Traffic Flows :**
   We consider a set of incoming traffic flows F, where each flow $f \in F$ has a forwarding policy (SFC) s_f, and a bandwidth demand $\hat{b}_f$. s_f represents the sequence of VNF types that the flow must be routed through. Here, we distinguish between three types of traffic flows: traffic originating from a host in a network and destined to a host in the same network, traffic originating from a host in the network and destined to a host outside the network, or traffic originating from and destined to a host outside the network. In this work, we consider that the ingress and egress nodes for incoming traffic flows are known apriori, denoted by n_f and n'_f, respectively. We represent a traffic flow request as a virtual graph $G^v_f = (V_f,E_f)$ where $V_f = \{n_f,s_f,n'_f\}$ represents the nodes in the virtual graph, and $E_f = \{(n_f,m_i), (m_i,m_{i+1}),...,(m_{|s_f|},n'_f)\}$ $(1 \leq i \leq |M|)$ represents the virtual links.

4. **The VNF Assignment Problem :**
   The VNF assignment problem consists of determining the optimal deployment of VNF instances that maximizes the amount of flows routed across the network. A flow can only be routed/admitted if its associated policy is met; that is, it is assigned a single instance of every VNF type in its forwarding policy, and successfully routed through these instances in the correct order. Figure 7.2(a) shows an example of a policy-aware traffic of a single flow across a substrate network. Here, we observe a flow with a chain of 3 VNF types, abstracted as a virtual graph of 5 nodes and 4 virtual links. The number next to

each virtual link denotes the bandwidth demand of the flow, while the digit next to each physical node/link denotes its capacity.

The VNF Assignment problem can be formally defined as follows:

Problem Definition 7.1. *Given a substrate $G^s = (N, E)$ and a set of flows F, each with a forwarding policy s_f, find the optimal placement of VNFs that maximizes the number of admitted traffic flows, while respecting the capacity constraints of the substrate network.*

Deploying VNF instances is highly correlated with the number of flows that can be admitted, and it entails not only determining the location/physical host of each instance, but also the number of instances needed of every VNF type. The location of a VNF instance affects the amount of bandwidth consumed to route traffic flows through this instance; particularly if the ingress/egress node of the flows are too far from the location of the deployed instance. To better illustrate this, consider the example in Figure 7.2; here, we observe that a different VNF placement for the same flow yields a different network resources utilization. In Figure 7.2(a), the location of the intermediate VNF in the flow's chain caused a loop/detour in the flow's path towards its egress node in order to respect the policy chain. Alternatively, a careful embedding of the VNFs (as shown in Figure 7.2(b)) greatly reduces the network's resource utilization, and subsequently its admissibility. Hence, to reduce the amount of bandwidth consumed, the network operator can decide to alternate the placement of the VNFs, or even deploy more VNF instances while incurring a higher cost in terms of substrate node resources needed to host the various VNFs. Clearly, the VNF assignment problem entails multiple intricate challenges, and can be logically divided into three sub-problems: (1) VNF instances deployment, which indicates the number and placement of VNF instances; (2) flow-to-VNF instance assignment; and (3) policy-aware routing for each flow along its designated VNF instances. Note that the flow-to-VNF instance assignment ensures that each admitted flow is assigned a single instance of every VNF type in its forwarding policy, and that the total load on any VNF instance does not exceed its processing capacity. Let H denote the set of all VNF assignment solutions. Each $h \in H$ represents a feasible solution for a set of incoming flows F; $h = (h_N, h_M, h_E)$ thus holds the solution for the following three subproblems:

- VNF Mapping: h_N: $M \longrightarrow N$.
- Flows-to-VNF assignment: h_M: $F \longrightarrow \bar{M}$. $\bar{M}$ represents the instantiated VNF types.
- Policy-Aware Traffic Routing: h_E: $F \longrightarrow R$. R represents the substrate paths used to route the flows in F; each composed of one or many substrate links.

7.2.2 Problem Formulation

In this section, we mathematically formulate the VNF assignment problem with the objective of maximizing the total number of flows admitted.

Parameters:

- $G^s(N, L)$: Substrate network with N nodes and L links.
- c_n: the capacity of substrate node n.
- $b_{i,j}$: the capacity of substrate link (i,j); where i and j denote the source and destination of the link, respectively.
- F: the set of flows, where every flow $f \in F$ is abstracted as a virtual graph $G_f^v(V_f, E_f)$, with a demand $\hat{b}_f$. Each virtual link $e \in E_f$ is composed of a pair of VNF types, where $o(e)$ and $d(e)$ denote the source and destination VNF types of edge e, respectively.
- M: the set of VNF types.
- p_m: the processing capacity of VNF type m.
- w_m: the resource demand of VNF type m.
- K_m: the maximum number of instances allowed for any VNF type m.

Decision Variables:

- $x_{m,n}^k = \begin{cases} 1, & \text{if instance } k \text{ of } m \text{ is placed on } n. \\ 0, & \text{otherwise} \end{cases}$
- $\delta_{m,n}^{f,k} = \begin{cases} 1, & \text{if } f \text{ is assigned instance } k \text{ of type } m \text{ on } n. \\ 0, & \text{otherwise} \end{cases}$
- $a_f = \begin{cases} 1, & \text{if flow } f \text{ is admitted.} \\ 0, & \text{otherwise.} \end{cases}$
- $y_{i,j}^{e,f} = \begin{cases} 1, & \text{if } e \in E_f \text{ is routed through link } (i,j). \\ 0, & \text{otherwise.} \end{cases}$
- $t_{i,j}$: indicates the amount of traffic measured on link (i,j).

- **Mathematical Model:**

$$Max \quad \sum_{f \in F} a_f \tag{7.1}$$

Subject to

VNF Placement

$$\sum_{n \in N} x_{m,n}^k \leq 1 \quad \forall 1 \leq k \leq K_m,\ m \in M \tag{7.2}$$

$$\sum_{k=1}^{K_m} \sum_{n \in N} x_{m,n}^k \leq K_m \quad \forall m \in M \tag{7.3}$$

$$\sum_{m \in M} \sum_{k=1}^{K_m} x_{m,n}^k . w_m \leq c_n \quad \forall n \in N \tag{7.4}$$

Flow-to-VNF Assignment

$$\delta_{m,n}^{f,k} \leq x_{m,n}^k \quad \forall 1 \leq k \leq K_m,\ m \in s_f,\ n \in N,\ f \in F \tag{7.5}$$

$$\sum_{n \in N} \sum_{k=1}^{K_m} \delta_{m,n}^{f,k} \leq 1 \quad \forall m \in s_f,\ f \in F \tag{7.6}$$

$$\sum_{f \in F} \sum_{n \in N} \delta_{m,n}^{f,k} . \hat{b}_f \leq p_m \quad \forall 1 \leq k \leq K_m, m \in M \tag{7.7}$$

$$x_{m,n}^k \leq \sum_{f \in F} \delta_{m,n}^{f,k} \quad \forall 1 \leq k \leq K_m,\ m \in M,\ n \in N \tag{7.8}$$

Policy-Aware Traffic Routing

$$\sum_{j:(i,j) \in L} y_{i,j}^{e,f} - \sum_{j:(j,i) \in L} y_{j,i}^{e,f} = \sum_{k=1}^{K_m} \delta_{o(e),n}^{f,k} - \sum_{k=1}^{K_m} \delta_{d(e),n}^{f,k} \quad \forall e \in E_f,\ f \in F, i \in N. \tag{7.9}$$

$$t_{i,j} = \sum_{f \in F} \sum_{e \in E_f} y_{i,j}^{e,f} . \hat{b}_f \quad \forall (i,j) \in L \tag{7.10}$$

$$t_{i,j} \leq b_{i,j} \quad \forall (i,j) \in L \tag{7.11}$$

$$a_f.|s_f| = \sum_{m \in s_f} \sum_{k=1}^{K_m} \sum_{n \in N} \delta_{m,n}^{f,k} \quad \forall f \in F \tag{7.12}$$

Our objective function aims to maximize the number of admitted flows. Constraint (7.2) avoids duplicate placement of any VNF instance. Constraint (7.3) ensures that the number of instances placed of any VNF type does not exceed the maximum number of instances allowed. Constraint (7.4) is the substrate nodes capacity constraint. Constraint (7.5) resolves the flow-to-VNF assignment, and Constraint (7.6) ensures that each flow can use at most one instance of any middlebox type. Constraint (7.7) ensures that the processing capacity of each VNF instance is not violated, while Constraint (7.8) makes sure that a VNF type will never be instantiated if is not assigned to at least a single flow. Constraint (7.9) is the flow conservation constraint to perform the policy-aware traffic routing, while Constraint (7.10) measures the traffic incurred on each substrate link, and Constraint (7.11) avoids substrate links capacity violation. Finally, Constraint (7.12) indicates that a flow is routed, if and only if, it is assigned a single instance of each VNF type in its corresponding policy.
The VNF assignment problem has been proven [147] to be NP-Hard via a reduction from the capacitated facility location problem. Hence in the following section, we provide a cut-and-solve based approach to exactly solve the problem within considerably much faster runtime than the ILP-based formulation.

7.3 A Cut-and-Solve Based Approach

In this section, we present our cut-and-solve based algorithm to provide an exact solution for the VNF assignment problem. Cut-and-solve [180] consists of decomposing the problem into two subproblems a master model and a subproblem. The master model is an ILP with a sparse search space, and hence is easier to solve than the original problem. At every iteration, a piercing cut is generated and added to the master to tighten its search space. Solving the master problem provides an upper bound to the original problem, while solving the subproblem provides a lower-bound. As the cuts accumulate, the search space will become tighter, and the cuts more constructive. When the upper-bound and lower-bound converge, then the obtained solution is optimal and the algorithm terminates. Figure 7.3 illustrates our proposed cut-and-solve based approach. First, a pre-processing is performed to tighten the bound of the master model, then the master and the subproblem are executed iteratively, where O is the number of flows admitted by the master, and P those admitted

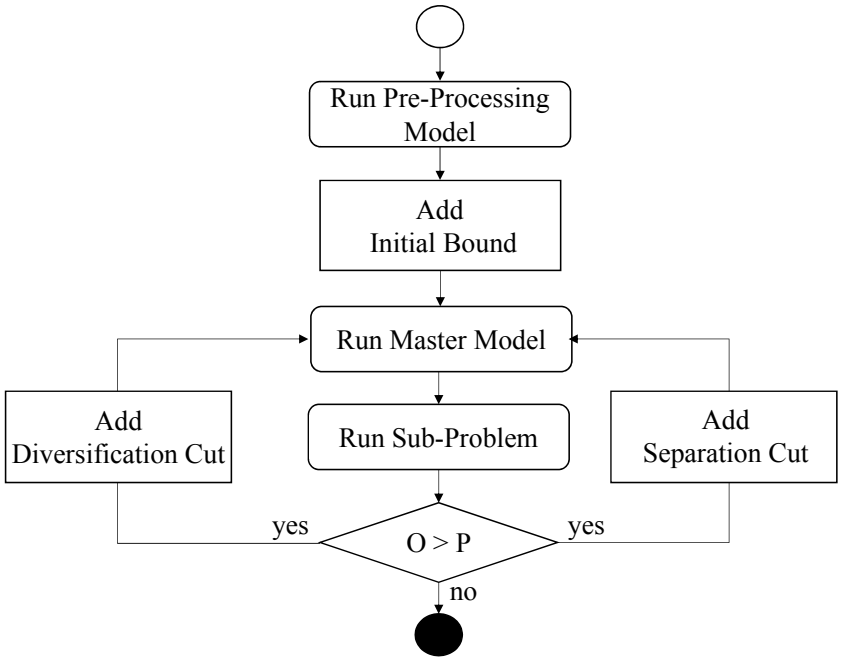


Figure 7.3: Cut-and-Solve Flow Chart

by the subproblem. At the end of every iteration, if the subproblem could not route all the flows admitted by the master ($P < O$), two classes of cuts are generated: a *diversification cut* extracted from the set of flows that were not admitted at every iteration, and a *separation cut* to avoid bottleneck links in the substrate network. The details of our cut-and-solve algorithm are provided below.

7.3.1 Pre-Processing Model

The pre-processing model consists of adding an initial cut on the master model's search space. This is achieved by running a Multi-Commodity Flow (MCF) problem that attempts to route each flow from its ingress to its egress node. The pre-processing model can be formulated as follows:

$$G \;=\; Max \;\; \sum_{f \in F} a_f \tag{7.13}$$

Subject to

$$\sum_{j:(i,j)\in L} y^f_{i,j} - \sum_{j:(j,i)\in L} y^f_{j,i} = \begin{cases} a_f, \; if \; i = n_f \\ -a_f, \; if \; i = n'_f \\ 0, \; otherwise. \end{cases} \tag{7.14}$$

$$\sum_{f \in F} y_{i,j}^{f}.\hat{b}_f \leq b_{i,j} \quad \forall (i,j) \in L \tag{7.15}$$

Constraint (7.14) ensures that each admitted ($a^f = \{0,1\}$) flow is routed ($y_{i,j}^f = \{0,1\}$) from its ingress to its egress node, and Constraint (7.15) is the substrate links capacity constraint. Solving the pre-processing thus provides an upper-bound G on the optimal solution that can be obtained by the master model; since if the substrate network's capacity can only accommodate a maximum of $G \subset F$ flows, then any policy-aware routing, that accounts for the VNFs placement, cannot exceed G due to the physical links capacities.

7.3.2 The Master Model

The master model consists of solving the VNF placement and flow-to-VNF assignment subproblems without any consideration to the substrate network's bandwidth capacity. This indeed makes the problem much easier to solve than the original problem. The solutions to the VNF placement and flow-to-VNF assignment subproblems are represented by two variables $s_{e,f}^n$ and $d_{e,f}^n$, which indicate the "physical" host of the source and destination of every virtual link. Given that each virtual link represents a pair of VNF types in a flow's chain, this therefore means that a VNF instance of the required VNF type is deployed on the source and destination hosts, respectively. Thus, each placed VNF instance is immediately associated with a particular flow; thereby, avoiding the placement of VNF instances that are not associated with any flow. The master model is mathematically formulated as follows:

Decision Variables:

- $x_{m,n}^k = \begin{cases} 1, & \text{if instance } k \text{ of } m \text{ is placed on } n. \\ 0, & \text{otherwise.} \end{cases}$
- $u_{e,f} = \begin{cases} 1, & \text{if virtual link } e \text{ of } f \text{ is admitted.} \\ 0, & \text{otherwise.} \end{cases}$
- $s_{e,f}^n = \begin{cases} 1, & \text{if } n \text{ is the source of edge } e \ (e \in E_f). \\ 0, & \text{otherwise.} \end{cases}$
- $d_{e,f}^n = \begin{cases} 1, & \text{if } n \text{ is the destination of } e \ (e \in E_f). \\ 0, & \text{otherwise.} \end{cases}$

- $q^n_{e1,e2} = \begin{cases} 1, & \text{if } e_1 \text{ and } e_2 \text{ meet at } n. \\ 0, & \text{otherwise.} \end{cases}$
- $a_f = \begin{cases} 1, & \text{if flow } f \text{ is admitted.} \\ 0, & \text{otherwise.} \end{cases}$
- **Mathematical Model:**

$$O \;=\; Max \sum_{f \in F} a_f \tag{7.16}$$

Subject to

$$\sum_{n \in N} s^n_{e,f} \leq a_f \quad \forall e \in \{E_f - e_1\},\ f \in F \tag{7.17}$$

$$\sum_{n \in N} d^n_{e,f} \leq a_f \quad \forall e \in \{E_f - e_{|E_f|}\},\ f \in F \tag{7.18}$$

$$u_{e,f} \leq (\sum_{n \in N} s^n_{e,f} + \sum_{n \in N} d^n_{e,f})\frac{1}{2} \quad \forall e \in E_f,\ f \in F \tag{7.19}$$

$$s^n_{e,f} \leq \sum_{k=1}^{K_m} x^k_{o(e),n} \quad \forall e \in E_f,\ f \in F,\ n \in N \tag{7.20}$$

$$d^n_{e,f} \leq \sum_{k=1}^{K_m} x^k_{d(e),n} \quad \forall e \in E_f,\ f \in F,\ n \in N \tag{7.21}$$

$$\begin{aligned} q^n_{e_i,e_{i+1}} \leq (d^n_{e_i,f} + s^n_{e_{i+1},f})\frac{1}{2} \quad & \forall (e_i, e_{i+1}) \in E_f : \\ & \{1 \leq i \leq |E_f| - 1\},\ f \in F,\ n \in N \end{aligned} \tag{7.22}$$

$$\sum_{n \in N} q^n_{e_i,e_{i+1}} \leq 1 \quad \forall (e_i, e_{i+1}) \in E_f : \{1 \leq i \leq |E_f| - 1\},\ f \in F \tag{7.23}$$

$$\sum_{n \in N} q^n_{e_i,e_{i+1}} \geq a_f \quad \forall (e_i, e_{i+1}) \in E_f : \{1 \leq i \leq |E_f| - 1\},\ f \in F \tag{7.24}$$

$$\sum_{n \in N} \sum_{k=1}^{K_m} x^k_{m,n} \leq K_m \quad \forall m \in M \tag{7.25}$$

$$\sum_{m \in M} \sum_{k=1}^{K_m} x_{m,n}^{k}.w_m \leq c_n \quad \forall n \in N \tag{7.26}$$

$$\sum_{f \in F} \sum_{e \in E_f : o(e)=m} s_{e,f}^{n}.\hat{b}_f \leq \sum_{k=1}^{K_m} x_{m,n}^{k}.p_m \ \forall m \in M, n \in N \tag{7.27}$$

$$a_f \leq u_{e,f} \quad \forall e \in E_f, \ f \in F \tag{7.28}$$

We let the objective function be to maximize the number of flows routed. Constraints (7.17) and (7.18) indicate that the source and destination of a virtual link can be placed on at most a single node each. Constraint (7.19) indicates that a virtual link is admitted, if and only if, both of its edge nodes have been assigned a substrate host. Constraints (7.20) and (7.21) ensure that the edge nodes of a virtual link cannot be placed on a given node if that nodes does not host the required VNF type. Constraints (7.22), (7.23), and (7.24) indicate that a flow can use at most a single instance of any VNF type, by making sure that every pair of contiguous virtual links share at least a single host. Constraint (7.25) makes sure that the maximum number of instances allowed for any VNF type is not violated. Constraint (7.26) is the substrate nodes capacity constraint. Constraint (7.27) is the VNF processing capacity constraint. Finally, Constraint (7.28) indicates whether a flow is routed.

7.3.3 The Subproblem Model

Every iteration of the master model is followed by running a subproblem to obtain a lower-bound. Given the source and destination of every virtual link ($\bar{s}_{e,f}^{n}$ and $\bar{d}_{e,f}^{n}$, respectively), the subproblem attempts to route the maximum number of flows possible without violating the substrate links capacity constraints. The subproblem model can thus be formulated as follows:

Parameters:

- $\bar{s}_{e,f}^{n} = \begin{cases} 1, & \text{if } n \text{ is the source of edge } e. \\ 0, & \text{otherwise.} \end{cases}$
- $\bar{d}_{e,f}^{n} = \begin{cases} 1, & \text{if } n \text{ is the destination of } e. \\ 0, & \text{otherwise.} \end{cases}$

Decision Variables:

- $a_f = \begin{cases} 1, & \text{if flow } f \text{ can be admitted.} \\ 0, & \text{otherwise.} \end{cases}$
- $z_{e,f} = \begin{cases} 1, & \text{if edge } e \text{ of flow } f \text{ is routed.} \\ 0, & \text{otherwise.} \end{cases}$
- $y_{i,j}^{e,f} = \begin{cases} 1, & \text{if edge } e \text{ of flow } f \text{ is routed through } (i,j). \\ 0, & \text{otherwise.} \end{cases}$
- **Mathematical Model:**

$$Max|E_f|.\sum_{f\in F} a_f + \sum_{e\in E_f}\sum_{f\in F} z_{e,f} \tag{7.29}$$

Subject to

$$\sum_{j:(i,j)\in L} y_{i,j}^{e,f} - \sum_{j:(j,i)\in L} y_{j,i}^{e,f} = a_f(\bar{s}_{e,f}^{i} - \bar{d}_{e,f}^{i}) \quad \forall i \in N, e \in E_f, f \in F \tag{7.30}$$

$$\sum_{f\in F}\sum_{e\in E_f} y_{i,j}^{e,f}.\hat{b}_f \leq b_{i,j} \quad \forall (i,j) \in L \tag{7.31}$$

$$z_{e,f} \geq y_{i,j}^{e,f} \quad \forall e \in E_f,\ f \in F,\ (i,j) \in L \tag{7.32}$$

$$z_{e,f} \leq \sum_{(i,j)\in L} y_{i,j}^{e,f} \quad \forall e \in E_f,\ f \in F \tag{7.33}$$

$$a_f.|E_f| = \sum_{e\in E_f} z_{e,f} \quad \forall f \in F \tag{7.34}$$

The subproblem's objective function is to maximize the number of flows admitted, as well as the total number of virtual links routed for every flow. The purpose of this dual objective is to encourage the subproblem to route the maximum number of flows possible (by adding a weight on the number of flows routed), while also indicating the virtual links that could be routed. The latter will be used to generate the piercing cuts on the virtual links that could not be routed. Constraint (7.30) represents the flow conservation constraint, while Constraint (7.31) measures the traffic routed through each physical link (i,j) and makes sure that the measured traffic does not exceed the capacity of the substrate link. Constraint

(7.32) and (7.33) indicate the virtual links routed, while Constraint (7.34) indicates that a flow is admitted, if and only if, all of its virtual links have been routed.

7.3.4 Piercing Cuts

As we have previously mentioned, at every iteration, two classes of cuts are generated: diversification cuts induced from the set of virtual links that could not be routed, and separation cuts to avoid bottleneck links.

1. **Diversification Cuts:** At the end of every iteration of the master, the subproblem is fed with a solution for the VNF placement and flow-to-VNF assignment subproblems. Subsequently, the subproblem attempts to route each virtual link across the substrate network. Let $\bar{F}$ be the set of flows admitted by the master, and $\hat{F} \subset \bar{F}$ be the subset of flows that the subproblem failed to admit; that is failed to route all of its virtual links. From this set of unrouted flows, a diversification cut can be induced to encourage the master to step away from this solution. This can be achieved by forcing the master to change the source or destination of at least one virtual link belonging to a flow in $\hat{F}$. To better illustrate this, consider the example in Figure 7.4 of a flow that was

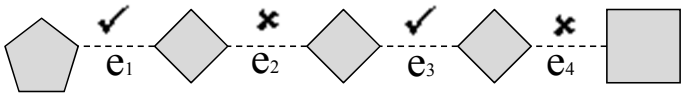


Figure 7.4: Cut Induced from Unrouted Virtual Links

admitted by the master but rejected by the subproblem. The reason why the flow was rejected is because virtual links e_2 and e_4 could not be routed. The failure to route virtual link e_2 can either be caused by the location selected for its source or that of its destination; similar observation can be said for e_4. Subsequently, the cut must force the master to change the source or destination of either e_2 or e_4; but not necessarily both because re-routing one virtual link may facilitate routing the other (in case of overlapping substrate links), and vice-versa. The proposed cut can be expressed as $s_{e_2} + d_{e_2} + s_{e_4} + d_{e_4} <= 3$. Note that in the case of two contiguous unrouted virtual links e_i and e_{i+1}, the cut will include the source of e_i and the source and destination of e_{i+1} (excluding the destination of e_i to avoid redundancy).
To generalize this, let $\hat{E}_f$ denote the unrouted virtual links for flow f, and D denote the total count of unrouted virtual links's source and/or destination at the end of a

subproblem iteration; where D can be computed as follows:

$$D = \sum_{f \in \hat{F}} \sum_{e \in \hat{E}_f} \sum_{n \in N} \bar{s}^n_{e,f} + \sum_{f \in \hat{F}} \sum_{e \in \hat{E}_f : \{(e+1) \notin \hat{E}_f\}} \sum_{n \in N} \bar{d}^n_{e,f} \tag{7.35}$$

Hence, the proposed cut becomes:

$$\sum_{f \in \hat{F}} \sum_{e \in \hat{E}_f} \sum_{n \in N} s^n_{e,f} + \sum_{f \in \hat{F}} \sum_{e \in \hat{E}_f : \{(e+1) \notin \hat{E}_f\}} \sum_{n \in N} d^n_{e,f} \leq D - 1 \tag{7.36}$$

2. **Separation Cuts:** In addition to the cuts induced from unrouted virtual links, we

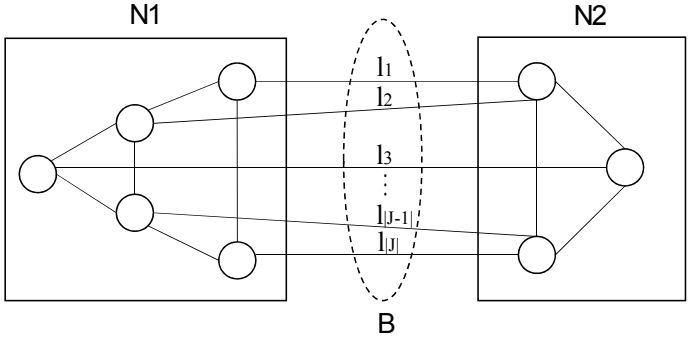


Figure 7.5: Separation Cut

also attempt to provide additional constructive cuts to the master to promote solutions that are more likely to yield a feasible link routing solution. To do so, for each unrouted flow, we try to detect the bottleneck link(s) that prevented the flow from being admitted by the subproblem. Subsequently, a separation cut is formulated by dividing the substrate network into two subsets N_1 and N_2; where N_1 includes the substrate nodes on one end of the bottleneck link(s) and N_2 the nodes on the other end as illustrated in Figure 7.5. The role of the separation cut is thus to ensure that the total amount of traffic routed from nodes in N_1 towards nodes in N_2 does not exceed the total capacity available on the physical links connecting N_1 and N_2. To generalize this, let B be the total amount of bandwidth capacity on the links connecting N_1 and N_2, a separation cut can be formulated as follows:

$$\sum_{f \in F} \sum_{e \in s_f} (s^{N_1}_{e,f}.d^{N_2}_{e,f} + d^{N_1}_{e,f}.s^{N_2}_{e,f}).\hat{b}_f \leq B \tag{7.37}$$

In the following section, we elucidate the algorithmic procedure for separation cuts generation.

7.4 Separation Cuts Generation

In this section, we provide the procedural details for the separation cuts generation (as shown in Algorithm 7.1). As we have previously mentioned, a separation cut is generated to encourage the master to step away from solutions that place the VNFs of a chain around bottleneck links. For each unrouted flow, a separation cut is generated around the bottleneck

Algorithm 7.1 Separation-Cuts-Generation Algorithm

Given:
$\hat{F}$ /*Set of Unrouted Flows by the Sub-problem*/
$S = \{\}$;
for each $(f \in \hat{F})$ **do**
    $N_1 = \{\}$;
    $N_2 = \{\}$;
    for each $(e \in E_f)$ **do**
        if (e is routed) **then**
            $N_1 = N_1 \cup s_e$;
            $N_1 = N_1 \cup d_e$;
        else
            $\bar{N} = RUN\text{-}BFS(s_e, \hat{b}_f)$;
            $N_1 = N_1 \cup \bar{N}$;
            $N_2 = N_2 \cup d_e$;
            Break
        end if
    end for
    $S_f = \{N_1, N_2\}$
    $S = S \cup S_f$;
end for
Return S;

links that prevented the routing of its virtual link(s). This is achieved as follows: for each unrouted flow, the virtual links composing the flow's chain are processed sequentially. If the virtual link has been successfully routed by the subproblem, its source and destination hosts (denoted as s_e and d_e respectively for each virtual link e) are placed in N_1. At the first occurrence of an unrouted virtual link, a Breadth First Search (BFS) is executed from the host of the virtual link's source. The aim of the BFS search is to explore the substrate nodes that are reachable from the source of the virtual link within the requested $\hat{b}_f$. Subsequently all of the reachable nodes, denoted as $\bar{N}$, are placed in N_1; and the host of the virtual link's destination in N_2. Next, all of the remaining substrate nodes (not in N_1) are added to N_2, and a separation cut is thus generated. To better illustrate this, consider the example presented

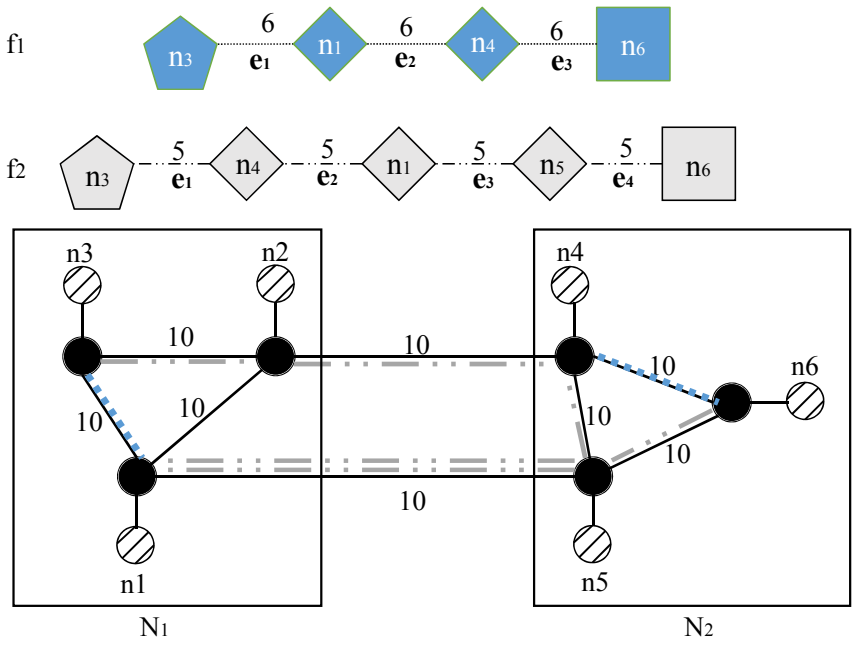


Figure 7.6: Example of a Separation Cut

in Figure 7.6. Here we observe two flows f_1 and f_2; f_1 demands 6 units of bandwidth, and its traffic must be routed from its ingress node n_3 to its egress node n_6, while being processed along the way by two VNF types. Similarly, f_2 demands 5 units of bandwidth, and its traffic must be routed from its ingress node n_3 to its egress node n_6, while being processed along the way by three VNF types. Following the execution of the master, f_1's VNF types are placed on nodes n_1 and n_4, while f_2's VNF types are placed on nodes n_4, n_1, and n_5. Subsequently, the subproblem is invoked to execute the virtual links routing; and we observe that f_2 was admitted by the subproblem, while f_1 was rejected due to the inability to route virtual link e_2. Here, Algorithm 1 is thus invoked to generate a separation cut for f_1. The algorithm begins by placing the physical source and destination of e_1 in N_1. Next, e_2 is detected as the first virtual link that could not be routed, and a BFS search is launched from n_1; One adjacent link of n_1 has enough capacity to reach n_2, but cannot move further than n_2; therefore n_2 is added to N_1, whereas n_4 and all other remaining facility nodes $\{n_5, n_6\}$ are placed in N_2. The generated separation cut thus indicates that the total sum of traffic that can be routed from nodes in N_1 to nodes in N_2 cannot exceed 20 (total bandwidth available on the substrate links interconnecting the two subsets). This would force the master to move the source of e_2 of flow f_1 to N_2, or its destination to N_1 to escape the bottleneck links in the next iteration.

The separation cut generation algorithm has a worst-case runtime of $O(|\hat{F}|.N^2)$. Note that

initial separation cuts can be generated by setting each facility node n in N_1, and the remaining facility nodes in $\{N\text{-}n\}$ in N_2. These initial cuts can be added at the pre-processing phase to prevent the master from assigning the source or destination of a subset of virtual links on substrate node n, if the total demand of these virtual links exceeds the total capacity of n's adjacent links.

7.5 Numerical Analysis

In this section, we numerically evaluate the performance of our cut-and solve based approach against the ILP model, as well as a heuristic-based approach that we denote as the "*k-ShortestPath*". The k-shortestPath heuristic is similar to Heuristic-A proposed in [174]; it consists of finding the k shortest paths between the ingress and egress node of every flow, and then attempts to place the VNF types associated with its policy along one of the generated paths. The generated paths are fed to an ILP model to decide on the optimal placement of VNFs that maximizes the number of flows admitted. In our numerical evaluation, we adopt two random network topologies R_1 with 40 nodes and 75 links, and R_2 with 60 nodes and 160 links, as well as a three-layered data center (DC) network topology of 36 nodes and 48 links. For each substrate network, we consider that each substrate node has a resource capacity of 48 CPU, and every link has a bandwidth capacity of 1 GBps. Further, the set of flows to be routed are randomly generated with bandwidth demand in the range [25-100] Mbps, and a random policy chain of up to 5 VNF types. The number of instances allowed for any VNF type is in the range between [10-20], with resource demand in the range [4,6,8,12] CPU, and a processing capacity of 1 GBps.

We decompose our numerical analysis into two sections: a performance analysis which consists of evaluating the cut-and-solve based approach against the ILP model; mainly by comparing the execution time (scalability) of the former compared to the latter, and a comparative analysis to compare the cut-and-solve against the k-shortestPath heuristic by looking at 4 main metrics: admission, runtime, revenue, and node and link utilization. All our test cases are performed on an 8GB RAM machine with a 3.6 GHz processing speed.

7.5.1 Performance Analysis

In this section, we evaluate the performance of our cut-and-solve based approach against the ILP model over the data center network topology. To do so, we look at the execution time of the ILP model against our proposed method as we vary the number of flows. The

Table 7.1: Runtime (sec) Analysis for Data Center Network ($N = 36$, $L = 48$)

# Flows	ILP Model		Cut-and-Solve	
	Admission	Runtime	Admission	Runtime
5	5	0.67	5	1.32
10	10	1.57	10	1.37
15	15	5.32	15	1.72
20	20	9.99	20	2.27
25	25	41.91	25	3.3
30	28	46.33	28	5.4
35	32	4506.32	32	6.42

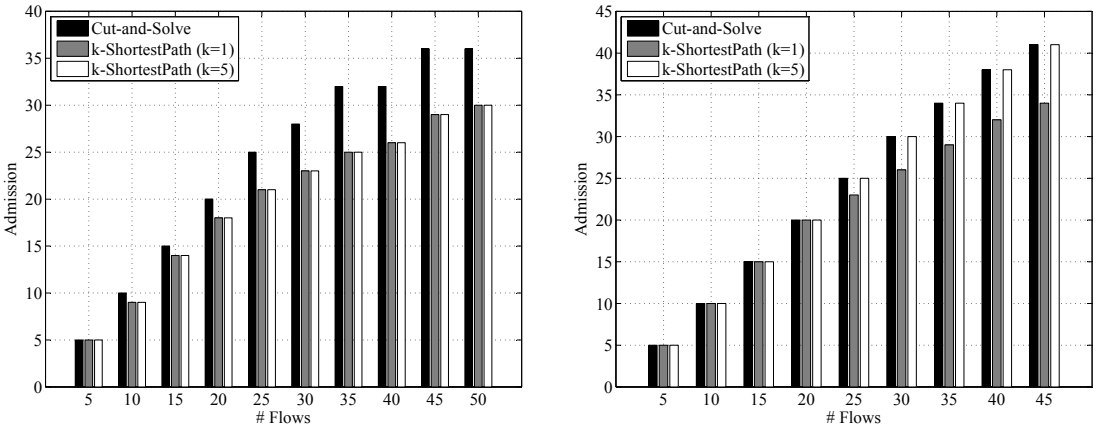
results are illustrated in Table 7.1. Clearly, we observe that the runtime of the ILP model grows exponentially. In fact, the ILP took 1 hour and 20 minutes to route 32 flows, while the cut-and-solve based approach was able to achieve the same admissibility in less than 7 seconds; that is 700 times faster than the ILP model. This clearly shows that the ILP-based approaches for solving this problem are not efficient and do not scale for larger instances.

7.5.2 Comparative Analysis

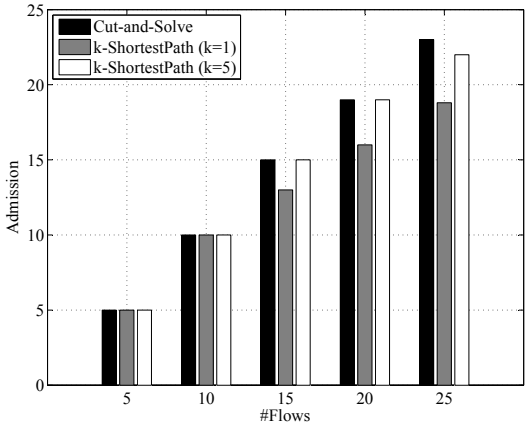
Now, we compare our cut-and-solve based approach against the k-ShortestPath algorithm for $k = 1$ and $k = 5$. Here, we look at 4 main metrics: runtime, admission, revenue, and node and link utilization.

- **Admission:**

  First, we look at the total admission achieved by our proposed approach against the k-ShortestPath heuristic. The results are illustrated in Figure 7.7. In all of the test cases, we observe that the cut-and-solve approach achieves a higher admission than the k-ShortestPath with k=1. For instance, to route 50 flows over the DC network (Figure 7(a)), we observe that the cut-and-solve approach achieves a 20% higher admission; even when the number of shortest path generated for the heuristic is increased to 5 (k-ShortestPath (k=5)). The multi-layered architecture of data center networks, although offers multiple paths between every pair of nodes, most of these paths share the same lower-layer links. For instance, in a FatTree network [28], each server rack is connected to the Top-of-rack switch by a single link. Subsequently, when the lower layer links are congested, no other alternative path will be able to route the traffic. Whereas for the random topologies (Figures 7(b)) and 7(c), we observe that increasing the number of shortest path generated to 5 enables the k-shortestPath algorithm to achieve comparable results to the optimal solution. However as the size of the problem increases



(a) DC (N=36,L=48) (b) R_1 (N=40,L=75)



(c) R_2 (N=60,L=160)

Figure 7.7: Admission

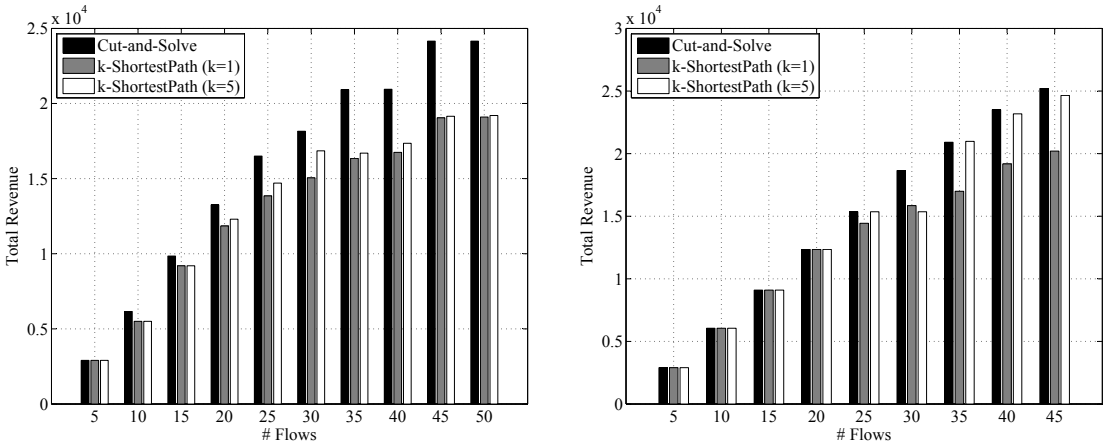
(as shown in Figure 7(c)), we observe that even for k set to 5, the k-shortestPath algorithm still falls short from attaining the optimal solution (obtained by the cut-and-solve). Further, as will be shown in the sequel, increasing the size of k comes at a considerable cost in terms of runtime.

- **Total Revenue:**

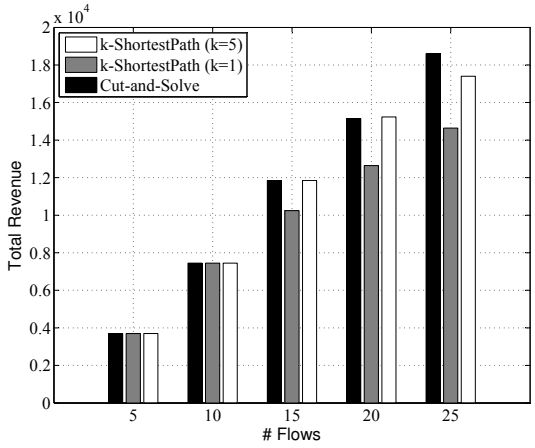
  Second, we look at total revenue achieved by each one of the evaluated methods (Figure 7.8). Revenue is measured as a function of the flow's demands in terms of bandwidth and the number of VNF types in its policy chain; which can be expressed as follows:

$$Revenue = \prod_{BW} b_f + \prod_{VNF} |s_f|. \tag{7.38}$$

  Here again, we observe that over the data center network, the cut-and-solve based approach achieves a much higher revenue than the k-ShortestPath algorithm for $k = 1$



(a) DC (N=36,L=48) (b) R_1 (N=40,L=75)



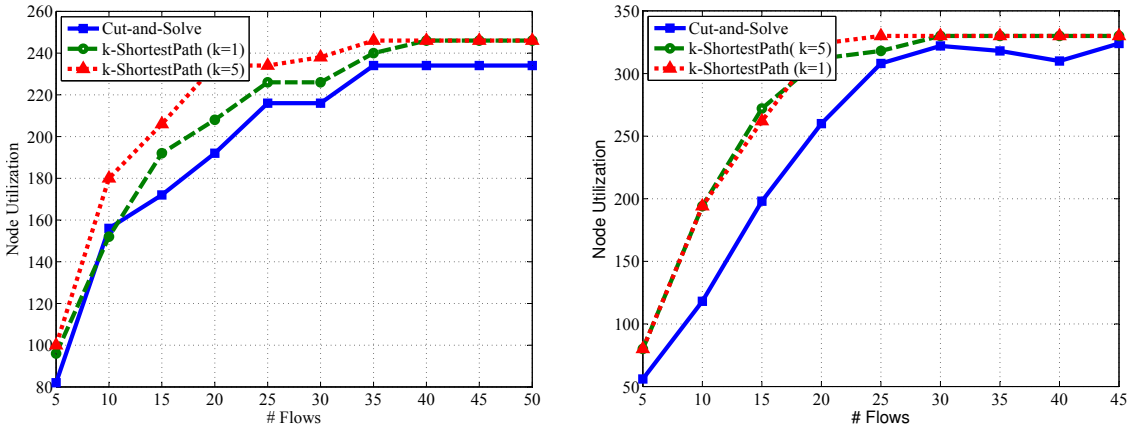
(c) R_2 (N=60,L=160)

Figure 7.8: Total Revenue

and $k = 5$. Indeed, when solving the VNF assignment problem for 50 flows over the DC network (Figure 8(a)), the cut-and-solve approach achieved 27% higher revenue than its peers. As for the random topologies (Figures 8(b) and 8(c)), we observe that the k-shortestPath algorithm with k=5 achieves comparable results to the optimal solution obtained by the cut-and-solve based approach; however, this comes at a huge cost in terms of runtime (as shown will be shown in Tables 7.2 and 7.3).

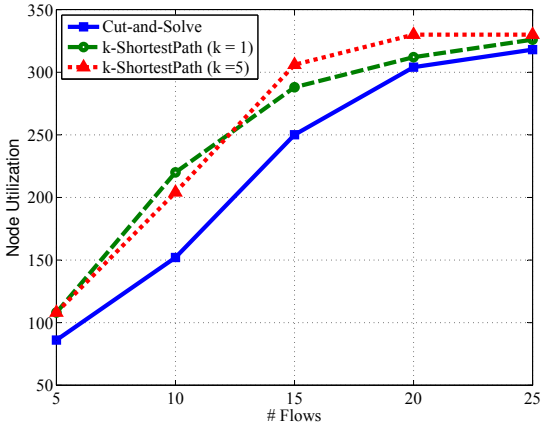
- **Node and Link Utilization:**

  Next, we look at the node and link utilization of the cut-and-solve approach against the k-ShortestPath Algorithm. The results are presented in Figures 7.9 and 7.10. In all of the adopted network topologies, we observe that the cut-and-solve based approach achieves a much lower node utilization. Hence, even when the k-shortestPath algorithm with k=5 achieves comparable admissibility (over R_1 and R_2), the incurred cost in



(a) DC (N=36,L=48)

(b) R_1 (N=40,L=75)

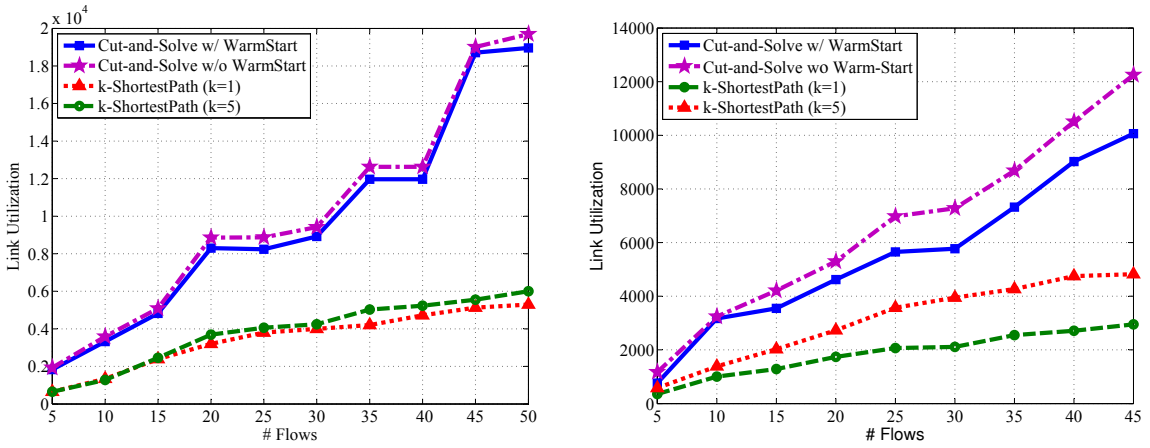


(c) R_2 (N=60,L=160)

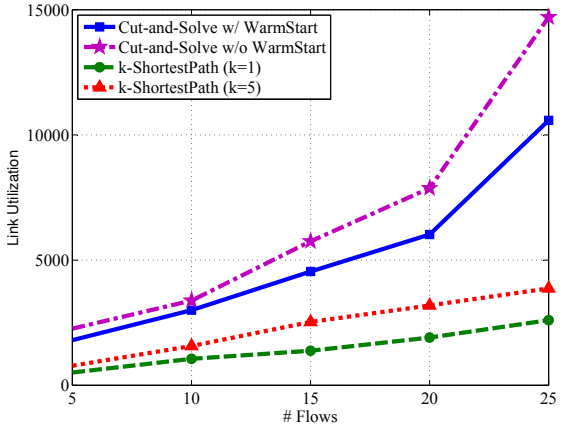
Figure 7.9: Node Utilization

terms of node utilization is much higher; which is expected since the k-shortestPath tackles the problem in a disjoint fashion by attempting to deploy VNFs on predefined paths.

Whereas for link utilization, we observe that the k-shortestPath algorithm achieves a much lower link utilization than the cut-and-solve based approach. To enhance the link utilization of the cut-and-solve based approach, at each iteration, we warm-start the subproblem by running a shortest-path algorithm (e.g. Dijkstra) and using the obtained results as an initial solution to the subproblem. Here, if the shortest-path algorithm was capable of routing all of the flows admitted by the master, then the subproblem will use the warm-start solution. Alternatively, it will try to enhance on the solution if possible. Clearly, the cut-and-solve based approach with warm-start achieves a much lower link utilization; for instance, over R_2 and for 30 flows, the warm-start enables the cut-and-solve based approach to achieve 27% less link utilization compared



(a) DC (N=36,L=48) (b) R_1 (N=40,L=75)



(c) R_2 (N=60,L=160)

Figure 7.10: Link Utilization

to the cut-and-solve without warm-start (as presented in Figure 10(c)). Note that the low node and link utilization achieved by the k-shortestPath with k=1 is a reflection of its low admissibility, and restricted input set (single shortest path).

- **Runtime** Finally, we look at how the runtime of the evaluated methods varies as we increase the number of flows. The results for random topology R_1 and R_2 are presented in Tables 7.2 and 7.3, respectively. We denote the cut-and-solve with and without warm-start as "CS w/ ws" and "CS w/o ws", respectively, and we denote the k-ShortestPath with k=1 and k=5 as "SP (k=1)" and "SP (k=5)", respectively. Here, we observe that runtime of all of the evaluated methods increases as we increase the number of flows. Mainly, we find that the warm-start slightly increases the runtime of the cut-and-solve based approach, since every iteration now entails also invoking the shortest-path algorithm. Second, we remark that the runtime of the k-shortestPath

Table 7.2: Runtime (sec) Analysis for $R_1(N = 40, L = 75)$

# Flows	CS w/ ws	CS w/o ws	SP (k=1)	SP (k=5)
5	0.94	0.76	0.04	0.17
10	2.02	1.59	0.08	0.24
15	3.22	2.30	0.09	0.34
20	4.68	3.32	0.09	0.59
25	7.92	5.72	0.17	1.05
30	13.42	9.66	0.58	17.33
35	31.4	26.10	1.05	195.97
40	46.12	39.63	1.91	567.24
45	59.3	50.61	3.93	6424.6

Table 7.3: Runtime (sec) Analysis for $R_2(N = 60, L = 160)$

# Flows	CS w/ ws	CS w/o ws	SP (k=1)	SP (k=5)
5	2.42	1.7	0.05	0.14
10	5.4	3.64	0.06	0.37
15	9.4	5.51	0.09	0.46
20	21	14.43	0.13	9.42
25	48	38.33	0.22	32484

algorithm is greatly affected by the number of paths generated. Namely, for R_2 and 20 flows, the k-shortestPath algorithm with k=5 is 72 times slower than the case where k=1. However, recall that generating more paths allows for higher admissability and achievable revenue. Finally, we observe that although the k-shortestPath with k=5 achieves comparable admissibility to the cut-and-solve over R_1 and R_2 network topologies; it is also highly unscalable. Namely, over R_2 and for 25 flows, it took 9 hours to return a solution that is at 4% gap from optimal (achieved by the cut-and-solve method as shown in Figure 7(c)).

As for the cut-and-solve based approach, we observe that its runtime also increases with the number of flows; yet in our test cases it remained less than a minute. For highly dynamic network traffic, our proposed approach may complement existing heuristic-based techniques by offering a timely approach that allows to evaluate the quality of the obtained solution, and can also be employed for periodic reconfiguration to enhance the substrate network's utilization over time.

7.6 Conclusion

In this chapter, we tackled the VNF-assignment problem. A prominent problem in the literature, where most of the existing contributions consisted of unscalable ILP-based approaches, or heuristics with no guarantee on the quality of the obtained solution. As opposed to existing work, we proposed a cut-and-solve based approach to solve exactly the VNF assignment problem. Our cut-and-solve method is equipped with two classes of piercing cuts that allow for quick convergence. We compared our proposed technique against the ILP formulation, as well as a heuristic-based method, and we have found that our approach can solve the VNF assignment problem to optimality within a considerably much faster runtime.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation is concerned with the problems of traffic and resource management in robust cloud data center networks. Both of these concerns are vital for enabling and supporting the multi-million dollar business of cloud computing. The traffic management concern is due to the unprecedented growth of the underlying infrastructure that runs cloud services, and the traffic shift from N/S to E/W. The resource allocation problem on the other hand is related to the virtualization of the cloud infrastructure that enabled better resource utilization due to consolidation, while also evoking the virtual-to-physical resource mapping problem; or what we referred to, throughout this manuscript, as the VNE problem.

We approached the problem of traffic management with policy-aware and policy-oblivious flows, where Chapter 2 addressed the latter by solving the NP-Complete VLAN assignment problem, and Chapter 7 tackled the former by studying the VNF assignment problem. As for the VNE problem, we introduced in Chapter 3 the VNE for services with one-to-many communication mode, and due to the failure prone nature of the cloud infrastructure, we dedicated Chapters 4, 5, and 6 to study the VNE with survivability and availability guarantees.

Namely in Chapter 2, we presented the traffic engineering problem in Layer-2 data center networks. Here, we showed that the use of multiple VLANs allows to overcome the limitations of the inherent traffic splitting technique of Layer-2 Ethernet switches. This lead us to define the NP-Complete VLAN assignment problem and elucidate on its exponential search space. Subsequently, we proposed an exact and a semi-heuristic decomposition to solve the VLAN assignment problem. Through numerical analysis we have shown that our proposed

decomposition explores less than 1% of the available search space, with an optimality gap of at most 4%.

In Chapter 3, we shifted our attention towards the resource allocation problem. By surveying the literature, we have shown that most of the existing work overlooked the mode of communication that cloud services may exhibit. Subsequently, in this dissertation, we considered the VNE problem for services with one-to-many communication mode, and presented the unique properties that differentiate multicast from one-to-one (unicast) services, as well as enlisted many real-life cloud services that can benefit from such a tailored embedding scheme. In this regard, we formally defined the multicast VNE problem and proved its NP-Hard nature in arbitrary graphs. We proposed a 3-Step approach for solving the MVNE problem, and proved that it can obtain optimal solution in polynomial time when employed for MVNs with homogeneous resource demands over tree-like DCN. For arbitrary graphs and MVNs with heterogeneous resource demands, we proposed a Tabu-based meta-heuristic for solving the MVNE problem. Through our numerical evaluation, we have shown that our Tabu-based MVNE achieves promising results over peer embedding techniques within encouraging runtime.

To ensure service continuity in failure-prone data center networks, in Chapters 4 and 5 we studied the survivable VNE problem against facility node failure. In Chapter 4, we considered the survivable VNE problem for unicast services, and we focused on the proactive scheme that consists of redesigning VNs into survivable VNs by augmenting the service with backup/redundant nodes. We showcased the limitation of existing literature that fix the number of backup nodes to a predefined constant value, and presented several motivational examples to support this claim. Further, we presented a novel prognostic redesign technique that we dubbed as "ProRed". ProRed's redesign not only finds the number of backup nodes needed, it also determines which subset of nodes each backup will protect, as well as its placement/location that promotes backup sharing. Via numerical analysis, we showcased the multiple gains that ProRed achieves compared to existing redesign techniques in the literature.

Chapter 5 also considered the survivable VNE problem; however it focused on the case of multicast services and proposed a reactive scheme to repair/restore multicast services in the event of facility node failure. Similarly to VNE, existing work on survivable VNE also overlooked the various mode of communication of cloud services. To this extent, this Chapter was devoted towards understanding the impact of failure on MVNs residing in cloud DCN. We formally defined the MVN restoration problem, and proved its NP-Complete nature.

Further, we proved that this problem can be solved in polynomial-time over tree-like DCN topologies.

Along the same efforts, we also considered the problem of providing a guarantee on service continuity by studying the "availability-aware" VNE problem. Here, the availability of the cloud service is the product of the availability of the physical servers hosting it. By surveying the existing literature on availability-aware VNE, we showed that existing work overlooked the problem of "availability-overprovisioning" which can greatly limit the admissibility of the substrate network. Further, the VN resource demands were considered static; as in their demands do not change overtime; which goes against the elastic nature of cloud services. To this extent, in Chapter 6 we proposed a novel framework that consists of two main modules: JENA an availability-aware embedding module for incoming services that provides "just-enough" availability, and ARES a reconfiguration module that manages the scale-up requests of hosted services over time while maintaining the "just-enough" availability guarantees. Our proposed framework presents encouraging gains over existing and benchmark schemes in terms of enhanced admissibility and long-term revenue.

Finally, in Chapter 7 we revisited the traffic management problem but this time we tackled the case of policy-aware traffic flows. We considered the problem of policy-aware traffic steering via "softwarized" network functions; commonly referred to as the NP-Complete VNF assignment problem. Here, most existing work consisted of ILP or heuristic based approaches; where the former lacks scalability and the latter provides no guarantees on the quality of the obtained solution. In this regard, we presented a cut-and-solve based approach for the VNF assignment problem which consists of decomposing the problem into two subproblems: a master and a subproblem. At every iteration, constructive piercing cuts are extracted from the subproblem and introduced to the master to tighten its search space until it converges to the optimal solution. Through our numerical evaluation, we showed that our approach achieves optimal solutions 700 times faster than ILP-based formulations.

8.2 Future Work

This thesis presented significant contributions in the areas of traffic and resource management for robust cloud data center networks. Yet, there exists several interesting future research directions that can be explored:

- For the proposed decomposition method presented in Chapter 2, and as we have previously mentioned, employing a more effective technique to go from the relaxed LP solution to the integral ILP solution can potentially improve our model's optimality gap. Hence, an interesting research direction can be to explore the impact of such techniques (e.g., branch-and-bound) to affirm this proclamation. Also, in our proposed work, we have assumed that the traffic demands are given. VLAN assignment problem with unknown traffic demands is a more challenging problem, thus another interesting future direction for this work is to devise an efficient online decomposition model that can handle the unpredictable nature of demands in cloud data centers. Another crucial point is to characterize traffic flows with more metrics, such as delay sensitivity, in order to broaden the scope of our current model.

- As for the MVNE problem presented in Chapter 3, given the dynamic nature of cloud services, an interesting future direction can be to propose a re-optimization technique to improve the network's resource utilization over time as MVNs leave the substrate network. Further to widen the scope of our suggested embedding technique, it would be interesting to study the problem of embedding VNs with heterogeneous communication, including the case where a single VN includes both unicast and multicast jobs.

- Now for the prognostic redesign proposed in Chapter 4, it would be interesting to explore the SVN design problem to handle multiple network components failure (which is rare but probable). Another important study can be to consider the SVN design problem for services with different mode of communication.

- For the problem of managing scaling requests of availability-aware services; recall that ARES only manages scale-up requests. Hence, and interesting extension for this work is to consider the problem of reconfiguring VN scale-down requests to optimize substrate network utilization efficiency over time, as well as the problem of reconfiguring VN scaling requests following bandwidth demands increase and/or arrival of new communication requirements.

- Finally, VNF is still in its infancy; and there exists multiple problems that need to be tackle to facilitate and enable its support. Some of the interesting research directions in this area is the availability-aware VNF assignment problem. Here, the availability of VNs is the product of the availability of the servers hosting its VMs, as well as the hosts of the various VNFs processing the communication demand between these VMs. The problem is also particularly more interesting in the case of stateless VNFs that are

shared by multiple tenants.

Another equally important problem is the problem of VNF scheduling. This problem arises in the case of stateless VNFs that are shared by multiple tenants. Here, the network provider must find the optimal way to schedule/organize the usage of different VNFs by different tenant flows. The problem is further aggravated when tenant's jobs are associated with deadlines and the network provider aims to achieve energy conservation by consolidating/processing tenants's jobs with the minimal number of VNFs. Another interesting factor that comes into play is managing the scheduling of VNFs for traffic flows for VNFs residing on distinct hosts (e.g., for fault-tolerance purposes). Here in addition to the time needed for each VNF to process the incoming flows, the transmission time to for from VNF to another must also be factored-in.

Bibliography

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[3] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. IEEE, 2009.

[4] Thijs Metsch, Andy Edmonds, et al. Open cloud computing interface infrastructure. In *Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN)*, 2010.

[5] A Andrieux, K Czajkowski, A Dan, K Keahey, H Ludwig, T Nakata, J Pruyne, J Rofrano, S Tuecke, and M Xu. Grid resource allocation agreement protocol wg (graap-wg), 2007.

[6] Nabil Bitar, Florin Balus, Marc Lasserre, Wim Henderickx, Ali Sajassi, Luyuan Fang, and Yuichi Ikejiri. Cloud networking: Framework and vpn applicability. *draft-bitar-datacenter-vpn-applicability (work in progress)*, 2011.

[7] Ieee launches pioneering cloud computing initiative. *[Online]. Available: http://standards.ieee.ord/news/2011/cloud.html*, 2011.

[8] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[9] Peter Mell and Tim Grance. The nist definition of cloud computing. *Communications of the ACM*, 53(6):50, 2010.

[10] Jeff Barr. Cloud computing, server utilization, & the environment. *[Online]. Available: https://aws.amazon.com/blogs/aws/cloud-computing-server-utilization-the-environment*, 2015.

[11] Rich Miller. Google uses about 900,000 servers. *[Online]. Available: http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers*, 2011.

[12] Richard Miller. Inside amazon's cloud computing infrastructure. *[Online]. Available: http://datacenterfrontier.com/inside-amazon-cloud-computing-infrastructure/*, 2015.

[13] Sebastien Anthony. Microsoft now has one million servers - less than google, but more than amazon, says ballmer. *[Online]. Available: http://www.extremetech.com/extreme/161772-microsoft-now-has-one-million-servers-less-than-google-but-more-than-amazon-says-ballmer*, 2013.

[14] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[15] Cisco's massively scalable data center. *[Online]. Available: http://www.cisco.com/c/dam/en/us/td/docs/solutions/Enterprise/Data_ Center/MSDC/1-0/MSDC_ Overview_ 1.pdf*, 2012.

[16] HP Networking Blog. Effects of virtualization and cloud computing on data center networks. *[Online]. Available: http://www.ics.uci.edu/ cs237/reading/Download.pdf*, 2011.

[17] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[18] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *NSDI*, pages 265–280, 2010.

[19] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[20] Ziyu Shao, Xin Jin, Wenjie Jiang, Minghua Chen, and Mung Chiang. Intra-data-center traffic engineering with ensemble routing. In *INFOCOM, 2013 Proceedings IEEE*, pages 2148–2156. IEEE, 2013.

[21] Mike Schlansker, Yoshio Turner, Jean Tourrilhes, and Alan Karp. Ensemble routing for datacenter networks. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, page 23. ACM, 2010.

[22] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 8. ACM, 2011.

[23] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 3–14. ACM, 2008.

[24] HO Viet, Yves Deville, Olivier Bonaventure, and Pierre Francois. Traffic engineering for multiple spanning tree protocol in large data centers. In *Teletraffic Congress (ITC), 2011 23rd International*, pages 23–30. IEEE, 2011.

[25] Yevgeniy Sverdlik. One minute of data center downtime costs us$7,900 on average. *[Online]. Available: http://www.datacenterdynamics.com/power-cooling/one-minute-of-data-center-downtime-costs-us7900-on-average/83956.fullarticle*, 2013.

[26] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361. ACM, 2011.

[27] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM, 2010.

[28] M Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Z Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network

virtualization: A survey. *Communications Surveys & Tutorials, IEEE*, 15(2):909–928, 2013.

[29] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 75–86. ACM, 2008.

[30] Albert Greenberg, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 57–62. ACM, 2008.

[31] Yang Liu, Jogesh K Muppala, Malathi Veeraraghavan, Dong Lin, and Mounir Hamdi. *Data center networks: Topologies, architectures and fault-tolerance characteristics*. Springer Science & Business Media, 2013.

[32] Bernard Golden. *Virtualization for dummies*. John Wiley & Sons, 2011.

[33] David G Andersen. Theoretical approaches to node assignment. *Computer Science Department*, page 86, 2002.

[34] Li Chen, Baochun Li, and Bo Li. Allocating bandwidth in datacenter networks: a survey. *Journal of Computer Science and Technology*, 29(5):910–917, 2014.

[35] Anath Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *Communications Surveys & Tutorials, IEEE*, 15(4):1888–1906, 2013.

[36] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. In *NSDI*, volume 3, pages 19–21, 2010.

[37] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H Low, and Lachlan LH Andrew. Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 233–244. ACM, 2011.

[38] Chuanxiong Guo, Guohan Lu, Helen J Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization

architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, page 15. ACM, 2010.

[39] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking (TON)*, 20(1):206–219, 2012.

[40] Yong Zhu and Mostafa H Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, volume 1200, pages 1–12, 2006.

[41] Patrick Wendell, Joe Wenjie Jiang, Michael J Freedman, and Jennifer Rexford. Donar: decentralized server selection for cloud services. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 231–242. ACM, 2010.

[42] Qi Zhang, Quanyan Zhu, Mohamed Faten Zhani, and Raouf Boutaba. Dynamic service placement in geographically distributed clouds. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 526–535. IEEE, 2012.

[43] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[44] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[45] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88. ACM, 2009.

[46] Zhiping Cai, Fang Liu, Nong Xiao, Qiang Liu, and Zhiying Wang. Virtual network embedding for evolving networks. In *GLOBECOM*, pages 1–5. IEEE, 2010.

[47] Uri Budnik. Lessons learned from recent cloud outages, 2013.

[48] Muntasir Raihan Rahman, Issam Aib, and Raouf Boutaba. Survivable virtual network embedding. In *NETWORKING 2010*, pages 40–52. Springer, 2010.

[49] Jielong Xu, Jian Tang, Kevin Kwiat, Weiyi Zhang, and Guoliang Xue. Survivable virtual infrastructure mapping in virtualized data centers. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 196–203. IEEE, 2012.

[50] Hongfang Yu, Vishal Anand, Chunming Qiao, and Hao Di. Migration based protection for virtual infrastructure survivability for link failure. In *Optical Fiber Communication Conference*, page OTuR2. Optical Society of America, 2011.

[51] Hongfang Yu, Chunming Qiao, Vishal Anand, Xin Liu, Hao Di, and Gang Sun. Survivable virtual infrastructure mapping in a federated computing and networking system under single regional failures. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE, 2010.

[52] Wai-Leong Yeow, Cédric Westphal, and Ulas C Kozat. Designing and embedding reliable virtual infrastructures. *ACM SIGCOMM Computer Communication Review*, 41(2):57–64, 2011.

[53] Hongfang Yu, Vishal Anand, Chunming Qiao, and Gang Sun. Cost efficient design of survivable virtual infrastructure to recover from facility node failures. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

[54] Bingli Guo, Chunming Qiao, Jianping Wang, Hongfang Yu, Yongxia Zuo, Juhao Li, Zhangyuan Chen, and Yongqi He. Survivable virtual network design and embedding to survive a facility node failure. *Lightwave Technology, Journal of*, 32(3):483–493, 2014.

[55] Qian Hu, Yang Wang, and Xiaojun Cao. Survivable network virtualization for single facility node failure: A network flow perspective. *Optical Switching and Networking*, 10(4):406–415, 2013.

[56] Jack Murphy and Thomas Ward Morgan. Availability, reliability, and survivability: An introduction and some contractual implications. *The Journal of Defense Software Engineering*, 2006.

[57] Amazon EC2. [Online]. Available: `http://aws.amazon.com/ec2/sla/`.

[58] Qi Zhang, Mohamed Faten Zhani, Maissa Jabri, and Raouf Boutaba. Venice: Reliable virtual data center embedding in clouds. In *INFOCOM, 2014 Proceedings IEEE*, pages 289–297. IEEE, 2014.

[59] Md Golam Rabbani, Mohamed Faten Zhani, and Raouf Boutaba. On achieving high survivability in virtualized data centers. *IEICE Transactions on Communications*, 97(1):10–18, 2014.

[60] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 171–180. IEEE, 2007.

[61] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, C Cui, H Deng, et al. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, pages 22–24, 2012.

[62] Vasek Chvatal. *Linear programming*. Macmillan, 1983.

[63] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[64] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 151–162. ACM, 2013.

[65] Wenfei Wu, Yoshio Turner, and Mike Schlansker. Routing optimization for ensemble routing. In *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, pages 97–98. IEEE Computer Society, 2011.

[66] S Skiena. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pages 225–227, 1990.

[67] Christian E Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[68] Mike McBride. Multicast in the data center overview. 2013.

[69] Yuting Miao, Qiang Yang, Chunming Wu, Ming Jiang, and Jinzhou Chen. Multicast virtual network mapping for supporting multiple description coding-based video applications. *Computer Networks*, 57(4):990–1002, 2013.

[70] Min Zhang, Chunming Wu, Ming Jiang, and Qiang Yang. Mapping multicast service-oriented virtual networks with delay and delay variation constraints. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[71] Amrit Iyer, Pranaw Kumar, and Vijay Mann. Avalanche: Data center multicast using software defined networking. In *Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on*, pages 1–8. IEEE, 2014.

[72] Dan Li, Yuanjie Li, Jianping Wu, Sen Su, and Jiangwei Yu. Esm: efficient and scalable data center multicast routing. *IEEE/ACM Transactions on Networking (TON)*, 20(3):944–955, 2012.

[73] Dan Li, Mingwei Xu, Ming-chen Zhao, Chuanxiong Guo, Yongguang Zhang, and Minyou Wu. Rdcm: Reliable data center multicast. In *INFOCOM, 2011 Proceedings IEEE*, pages 56–60. IEEE, 2011.

[74] Dan Liao, Gang Sun, Vishal Anand, and Hongfang Yu. Efficient provisioning for multicast virtual network under single regional failure in cloud-based datacenters. *TIIS*, 8(7):2325–2349, 2014.

[75] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 50–61. ACM, 2011.

[76] Pi-Rong Sheu and Shan-Tai Chen. A fast and efficient heuristic algorithm for the delay-and delay variation-bounded multicast tree problem. *Computer Communications*, 25(8):825–833, 2002.

[77] Dan Liao, Gang Sun, Vishal Anand, and Hongfang Yu. Survivable provisioning for multicast service oriented virtual network requests in cloud-based data centers. *Optical Switching and Networking*, 14:260–273, 2014.

[78] Xiaozhou Li and Michael J Freedman. Scaling ip multicast on datacenter topologies. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 61–72. ACM, 2013.

[79] Dan Li, Mingwei Xu, Ying Liu, Xia Xie, Yong Cui, Jingyi Wang, and Guihai Chen. Reliable multicast in data center networks. *Computers, IEEE Transactions on*, 63(8):2011–2024, 2014.

[80] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[81] Niels LM Van Adrichem, Christian Doerr, and Fernando A Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–8. IEEE, 2014.

[82] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6. IEEE, 2012.

[83] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. Opentm: traffic matrix estimator for openflow networks. In *Passive and active measurement*, pages 201–210. Springer, 2010.

[84] Kévin Phemius and Mathieu Bouet. Monitoring latency with openflow. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 122–125. IEEE, 2013.

[85] Benoit Claise. Cisco systems netflow services export version 9. 2004.

[86] Mea Wang, Baochun Li, and Zongpeng Li. sflow: Towards resource-efficient and agile service federation in service overlay networks. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 628–635. IEEE, 2004.

[87] Shubhajit Roy Chowdhury, M Faizul Bari, Rizwan Ahmed, and Raouf Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[88] Kai Chen, Chengchen Hu, Xin Zhang, Kai Zheng, Yan Chen, and Athanasios V Vasilakos. Survey on routing in data centers: insights and future directions. *Network, IEEE*, 25(4):6–10, 2011.

[89] Jiaxin Cao, Chuanxiong Guo, Guohan Lu, Yongqiang Xiong, Yixin Zheng, Yongguang Zhang, Yibo Zhu, Chen Chen, and Ye Tian. Datacast: a scalable and efficient reliable

group data delivery service for data centers. *Selected Areas in Communications, IEEE Journal on*, 31(12):2632–2645, 2013.

[90] Tatsuhiro Chiba, Mathijs den Burger, Thilo Kielmann, and Satoshi Matsuoka. Dynamic load-balanced multicast for data-intensive applications on clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 5–14. IEEE, 2010.

[91] Ramon Garcia Barbera. Special clouds for special needs: High performance computing clouds, 2012.

[92] Abhishek Gupta, Laxmikant V Kale, Filippo Gioachin, Verdi March, Chun Hui Suen, Bu-Sung Lee, Paolo Faraboschi, Richard Kaufmann, and Dejan Milojicic. The who, what, why and how of high performance computing applications in the cloud. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, 2013.

[93] Christian Vecchiola, Suraj Pandey, and Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 4–16. IEEE, 2009.

[94] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE, 2010.

[95] James Staten. Justifying your cloud investment: high-performance computing, 2011.

[96] Alex Sutton. New high performance capabilities for windows azure, 2014.

[97] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *Micro, Ieee*, 23(2):22–28, 2003.

[98] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[99] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[100] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.

[101] Pierluigi Crescenzi, Viggo Kann, Magnús Halldórsson, Marek Karpinski, and Gerhard Woeginger. A compendium of np optimization problems. *[Online.] Available: http://www.nada.kth.se/˜ viggo/problemlist/compendium. html*, 1997.

[102] Giorgio Ausiello, Vincenzo Bonifaci, Stefano Leonardi, and Alberto Marchetti-Spaccamela. Prize-collecting traveling salesman and related problems, 2007.

[103] Fred Glover and Manuel Laguna. *Tabu SearchâĹŮ*. Springer, 2013.

[104] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[105] JR Raphael. The worst cloud outages of 2013 (so far). *InfoWorld, July*, 2013.

[106] Joseph Tsidulko. The 10 biggest cloud outages of 2014 (so far). *[Online] Available: http://www.crn.com/slide-shows/cloud/300073433/the-10-biggest-cloud-outages-of-2014-so-far.htm/pgno/0/4*, 2014.

[107] Shihabur Rahman Chowdhury, Reaz Ahmed, Md Mashrur Alam Khan, Nashid Shahriar, Raouf Boutaba, Jeebak Mitra, and Feng Zeng. Protecting virtual networks with drone.

[108] vrealize suite. 2015.

[109] Open stack. *[Online] Available: https://www.openstack.org/*, 2015.

[110] Cloud Computing Partners. The importance of cloud management platforms for multicloud deployments. *[Online] Available: http://www.cloudtp.com/insights/white-papers/the-importance-of-cloud-management-platforms-for-multicloud-deployments/*, 2015.

[111] Hao Di, Vishal Anand, Hongfang Yu, Lemin Li, Dan Liao, and Gang Sun. Design of reliable virtual infrastructure using local protection. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 63–67. IEEE, 2014.

[112] Hao Di, Vishal Anand, Hongfang Yu, Lemin Li, Binhong Dong, and Qingrui Meng. Reliable virtual infrastructure mapping with efficient resource sharing. In *Communications, Circuits and Systems (ICCCAS), 2013 International Conference on*, volume 1, pages 198–202. IEEE, 2013.

[113] M Rizwanur Rahman and Raouf Boutaba. Svne: Survivable virtual network embedding algorithms for network virtualization. *Network and Service Management, IEEE Transactions on*, 10(2):105–118, 2013.

[114] Tao Guo, Ning Wang, Klaus Moessner, and Rahim Tafazolli. Shared backup network provision for virtual network embedding. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5. IEEE, 2011.

[115] Carlos Colman Meixner, Ferhat Dikbiyik, Massimo Tornatore, C Chuah, and Biswanath Mukherjee. Disaster-resilient virtual-network mapping and adaptation in optical networks. In *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, pages 107–112. IEEE, 2013.

[116] Sandra Herker, Ashiq Khan, and Xueli An. Survey on survivable virtual network embedding problem and solutions. In *ICNS 2013, The Ninth International Conference on Networking and Services*, pages 99–104, 2013.

[117] Ankit Singla, Chi-Yao Hong, Lucian Popa, and Philip Brighten Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, volume 12, pages 17–17, 2012.

[118] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 346–353. IEEE, 2001.

[119] Aiguo Fei, Junhong Cui, Mario Gerla, and Dirceu Cavendish. A âĂIJdual-treeâĂİ scheme for fault-tolerant multicast. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 3, pages 690–694. IEEE, 2001.

[120] David Breitgand, Gilad Kutiel, and Danny Raz. Cost-aware live migration of services in the cloud. In *SYSTOR*, 2010.

[121] Mohamed Faten Zhani, Qi Zhang, Gael Simon, and Raouf Boutaba. Vdc planner: Dynamic migration-aware virtual data center embedding for clouds. In *Integrated*

Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pages 18–25. IEEE, 2013.

[122] Frank K Hwang, Dana S Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992.

[123] Ajay Gulati, Ganesha Shanmuganathan, Anne M Holler, and Irfan Ahmad. Cloud scale resource management: Challenges and techniques. *HotCloud*, 11:3–3, 2011.

[124] Amip Shah, Cullen Bash, Ratnesh Sharma, Tom Christian, Brian J Watson, and Chandrakant Patel. The environmental footprint of data centers. In *ASME 2009 InterPACK Conference collocated with the ASME 2009 Summer Heat Transfer Conference and the ASME 2009 3rd International Conference on Energy Sustainability*, pages 653–662. American Society of Mechanical Engineers, 2009.

[125] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.

[126] Ramendra K Sahoo, Mark S Squillante, Anand Sivasubramaniam, and Yanyong Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Dependable Systems and Networks, 2004 International Conference on*, pages 772–781. IEEE, 2004.

[127] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[128] Di Niu, Hong Xu, Baochun Li, and Shuqiao Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *INFOCOM, 2012 Proceedings IEEE*, pages 460–468. IEEE, 2012.

[129] Sara Ayoubi, Yiheng Chen, and Chadi Assi. Pro-red: Prognostic redesign of survivable virtual networks for cloud data centers. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–9. IEEE, 2014.

[130] Peter Bodík, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A Maltz, and Ion Stoica. Surviving failures in bandwidth-constrained datacenters. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 431–442. ACM, 2012.

[131] Cheng Zuo, Hongfang Yu, and Vishal Anand. Reliability-aware virtual data center embedding. In *Reliable Networks Design and Modeling (RNDM), 2014 6th International Workshop on*, pages 151–157. IEEE, 2014.

[132] Yukio Ogawa, Go Hasegawa, and Masayuki Murata. Virtual network allocation for fault tolerance with bandwidth efficiency in a multi-tenant data center. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 555–562. IEEE, 2014.

[133] Xin Li and Chen Qian. Traffic and failure aware vm placement for multi-tenant cloud computing. In *Proceedings of IEEE IWQoS*, 2015.

[134] Houda Jmila, Ines Houidi, and Djamal Zeghlache. Rsforevn: Node reallocation algorithm for virtual networks adaptation. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–7. IEEE, 2014.

[135] Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Filip De Turck, and Steven Latré. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[136] Zichuan Xu, Weifa Liang, and Qiufen Xia. Efficient virtual network embedding via exploring periodic resource demands. In *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pages 90–98. IEEE, 2014.

[137] Ye Zhou, Xu Yang, Yong Li, Depeng Jin, Li Su, and Lieguang Zeng. Incremental re-embedding scheme for evolving virtual network requests. *Communications Letters, IEEE*, 17(5):1016–1019, 2013.

[138] Gang Sun, Vishal Anand, Hong-Fang Yu, Dan Liao, Yanyang Cai, et al. Adaptive provisioning for evolving virtual network request in cloud-based datacenters. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 1617–1622. IEEE, 2012.

[139] Steven S Skiena. *The algorithm design manual: Text*, volume 1. Springer Science & Business Media, 1998.

[140] Mung Chiang. *Geometric programming for communication systems*. Now Publishers Inc, 2005.

[141] Deze Zeng, Lin Gu, and Song Guo. *Cloud Networking for Big Data*. Springer, 2016.

[142] Chi-Guhn Lee and Zhong Ma. The generalized quadratic assignment problem. *Research Rep., Dept., Mechanical Industrial Eng., Univ. Toronto, Canada*, page M5S, 2004.

[143] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[144] Aaron Gember, Robert Grandl, Ashok Anand, Theophilus Benson, and Aditya Akella. Stratos: Virtual middleboxes as first-class entities. *UW-Madison TR1771*, 2012.

[145] Brian Carpenter and Scott Brim. Middleboxes: Taxonomy and issues. Technical report, 2002.

[146] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Gregoire Lefebvre, Ravi Manghirmalani, Ravishankar Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. Steering: A software-defined networking for inline service chaining. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

[147] Md Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, et al. Orchestrating virtualized network functions. *arXiv preprint arXiv:1503.06377*, 2015.

[148] Justine Sherry, Sylvia Ratnasamy, and Justine Sherry At. A survey of enterprise middlebox deployments. 2012.

[149] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[150] P. Quinn and T Nadeau. Service function chaining problem statement. *draft-ietf-sfc-problem-statement-07 (work in progress)*, 2014.

[151] S. Kumar et al. Service function chaining use cases in data centers. *draft-ietf-sfc-dc-use-cases-03*, 2015.

[152] W. Haeffner et al. Service function chaining use cases in mobile networks. *draft-ietf-sfc-use-case-mobility-05*, 2015.

[153] C. Xie et al. Service function chain use cases in broadband. *draft-meng-sfc-broadband-usecases-04*, 2015.

[154] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473. USENIX Association, 2014.

[155] Ye Yu, Qian Chen, and Xin Li. Distributed collaborative monitoring in software defined networks. *arXiv preprint arXiv:1403.8008*, 2014.

[156] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, and Xiang Zhang. Network function virtualization in the multi-tenant cloud. *Network, IEEE*, 29(3):42–47, 2015.

[157] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 323–336, 2012.

[158] Elisa Maini and Antonio Manzalini. Management and orchestration of virtualized network functions. In *Monitoring and Securing Virtualized Networks and Services*, pages 52–56. Springer, 2014.

[159] K Giotis, Y Kryftis, and V Maglaris. Policy-based orchestration of nfv services in software-defined networks. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5. IEEE, 2015.

[160] Stuart Clayman, Elisa Maini, Alex Galis, Antonio Manzalini, and Nicola Mazzocca. The dynamic placement of virtual network functions. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[161] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 163–174. ACM, 2014.

[162] Timothy Wood, KK Ramakrishnan, Jinho Hwang, Grace Liu, and Wei Zhang. Toward a software-based network: integrating software defined networking and network function virtualization. *Network, IEEE*, 29(3):36–41, 2015.

[163] Josep Batalle, Jordi Ferrer Riera, Eduard Escalona, and Joan Antoni Garcia-Espin. On the implementation of nfv over an openflow infrastructure: Routing function virtualization. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–6. IEEE, 2013.

[164] Ming Xia, Meral Shirazipour, Ying Zhang, Howard Green, and Attila Takacs. Network function placement for nfv chaining in packet/optical datacenters. *Journal of Lightwave Technology*, 33(8):1565–1570, 2014.

[165] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. Near optimal placement of virtual network functions. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1346–1354. IEEE, 2015.

[166] Mathieu Bouet, Jérémie Leguay, Théo Combe, and Vania Conan. Cost-based placement of vdpi functions in nfv infrastructures. *International Journal of Network Management*, 25(6):490–506, 2015.

[167] Po-Wen Chi, Yu-Cheng Huang, and Chin-Laung Lei. Efficient nfv deployment in data center networks. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5290–5295. IEEE, 2015.

[168] Abhishek Gupta, M Farhan Habib, Pulak Chowdhury, Massimo Tornatore, and Biswanath Mukherjee. Joint virtual network function placement and routing of traffic in operator networks. 2015.

[169] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 171–177. IEEE, 2015.

[170] Hendrik Moens and Filip De Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 418–423. IEEE, 2014.

[171] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 98–106. IEEE, 2015.

[172] Rashid Mijumbi. Placement and scheduling of functions in network function virtualization. *arXiv preprint arXiv:1512.00217*, 2015.

[173] Xin Li and Chen Qian. The virtual network function placement problem. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 69–70. IEEE, 2015.

[174] Ali Mohammadkhan, Sheida Ghapani, Guyue Liu, Wei Zhang, KK Ramakrishnan, and Timothy Wood. Virtual function placement and traffic steering in flexible and dynamic software defined networks. In *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*, pages 1–6. IEEE, 2015.

[175] Tamás Lukovszki, Matthias Rost, and Stefan Schmid. It's a match!: Near-optimal and incremental middlebox deployment. *ACM SIGCOMM Computer Communication Review*, 46(1):30–36, 2016.

[176] Li Erran Li, Vahid Liaghat, Hongze Zhao, MohammadTaghi Hajiaghay, Dan Li, Gordon Wilfong, Y Richard Yang, and Chuanxiong Guo. Pace: policy-aware application cloud embedding. In *INFOCOM, 2013 Proceedings IEEE*, pages 638–646. IEEE, 2013.

[177] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 27–38. ACM, 2013.

[178] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 19–24. ACM, 2013.

[179] James W Anderson, Ryan Braud, Rishi Kapoor, George Porter, and Amin Vahdat. xomb: extensible open middleboxes with commodity servers. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 49–60. ACM, 2012.

[180] Sharlee Climer and Weixiong Zhang. Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence*, 170(8):714–738, 2006.

Chapter 9

Appendix

9.1 Enumerating All Spanning Trees in a FatTree Network

The FatTree network consists of a multi-rooted tree-like topology built out of 3 layers of K-port switches, that connect $\frac{K^3}{4}$ Server Racks (SR). It consists of K pods, where each pod hosts 2 layers of switches: $\frac{K}{2}$ Aggregate Switches (AS) connected to $\frac{K}{2}$ Top of Rack (ToR) switches, forming a complete bipartite graph inside every pod. Further, each $\frac{K}{2}$ ToR is connected to $\frac{K}{2}$ SRs, and each $\frac{K}{2}$ ASs is connected to $\frac{K}{2}$ Core Switches (CS). Our spanning trees enumeration for the FatTree network is performed under the following constraint:

Constraint for Loop-Free Trees: *A packet that is sent upwards from a source, can only change its direction once to go downwards towards recipient nodes.*

Under this constraint, it has been proven [19] that any distribution tree constructed in the FatTree network will always be loop free. This constraint implies that starting at a particular source node, this node has a choice of a single ToR to forward the traffic upwards into the pod. Next, the ToR has a choice of $\frac{k}{2}$ ASs to forward the traffic to. The selected ASs will then need to either move the traffic upwards to CSs, or downwards to ToRs, or both. Each AS has the choice of $\frac{k}{2}$ - 1 ToRs and $k/2$ CSs as shown in Figure 5.4. We denote L_c as the number of ways a ToR switch goes up to n ASs, where only c of them are allowed to leave the pod. L_c can be formulated as follows:

$$L_c = \sum_{i=c}^{\frac{k}{2}} \binom{\frac{k}{2}}{i} \binom{i}{c} \sum_{j=0}^{c} \binom{c}{j} D(i-c+j, \frac{k}{2}-1) \tag{9.1}$$

$D(n,m)$ represents the number of ways m ToRs can be connected to n ASs, where each AS is connected to at least a single ToR and not two ASs are connected to the same ToR. $D(n,m)$ is a recurrence that can be computed as follows:

$$D(n,m) = \sum_{j=1}^{m-(n-1)} \binom{m}{j} D(n-1, m-j) \tag{9.2}$$

; such that :

$$D(n,m) = \begin{cases} 0, n = 0 \;\; || \;\; n > m \\ 1, n = 1 \end{cases} \tag{9.3}$$

Now, for a given L_c, c ASs are designated to leave the pod, that is each one of these ASs must select at least one CS out of its corresponding $\frac{k}{2}$ CSs. Note that each AS in a given pod is connected to distinct $\frac{k}{2}$ CSs. The selected CSs must span all the remaining k-1 pods. Assume the case where c ASs where designated to channel the traffic upwards, then the number of ways these c ASs can pick among their $\frac{k}{2}$ CSs to span the remaining k-1 pods (to reach other SRs) can be computed using the following recurrence:

$$D'(c, k-1) = \sum_{j=1}^{\frac{k}{2}} \binom{\frac{k}{2}}{j} \sum_{i=j}^{k-c} \binom{k-1}{i} D(j,i) D'(c-1, k-1-i) \tag{9.4}$$

Note that here again, we can make use of our recurrence $D(n,m)$ presented in Equation 9.4 to compute the number of ways m pods can be distributed to n CSs, where each CS is connected to at least one pod, and not two CSs are connected to the same pod.
Overall, the total number of spanning trees in a FatTree network is equal to $\sum_{c=1}^{\frac{k}{2}} L_c D'_c$.

9.2 JENA: VNE with Just-Enough Availability

Tabu is a widely adopted meta-heuristic algorithm, which was proven capable of achieving optimal and near-optimal solutions for various optimization problems. The most attractive feature of Tabu-based search is the use of "adaptive memory" to direct the search towards solutions that best service the desired objective function. In this work, we adopt the tabu-based search to find node mapping solution for every incoming VN that minimizes the amount of excess availability. In what follows, we present the various components of our Tabu-based approach for solving the availability-aware VNE.

1. **Initial Bare-Bone Solution:** Our initial solution is obtained via a greedy embedding heuristic, by mapping the VMs on the feasible physical nodes with the highest availability. If the greedy embedding returns a feasible solution with excessive availability, then the Tabu search is launched in the aim to find the lowest-cost embedding solution with "just-enough" availability. However, if the greedy placement failed to find an embedding solution that meets the requested availability demand, then this VN must be augmented with additional protection domains. Given the selection policy dictated by the cloud provider, the VN is augmented iteratively with a backup node, that is placed on the first vacant host with the highest availability. At every iteration, the availability of the VN is evaluated again. Once the availability of the initial greedy embedding solution surpasses the acceptable threshold, the algorithm proceeds to enhance on the solution found. Note that if no greedy embedding solution can be found due to resource scarcity, the VN request is rejected.

2. **The Neighborhood Structure:** We define a move $m^v_{n,n'}$ to be a shift of a virtual node v from one physical node n to another n'. Each virtual node v finds a set of candidate moves C_v by exploring the physical nodes within k hops from its current host location. While selecting the set of candidate moves, an active pruning is employed where candidate moves are automatically discarded if they fail to satisfy the resource requirement of v, or drop the overall availability of the virtual network below the requested value. The union of all candidate moves for the given virtual network is denoted as C. Next, the cost of each candidate move $m \in C$ is evaluated, using the following equation:

$$Cost_m = L + P_1 \tag{9.5}$$

   where L represents the sum of the load on each substrate host. Here, choosing the move with the least cost aims to encourage solutions that balance the load across the network, and hence on the long run increase the network's admissibility. Next, P_1 represents the penalty function for excessive availability. It consists of multiplying every unit of excess availability by 100. This allows Tabu to converge towards a solution that offers "just-enough" availability.

3. **Tabu Moves:** We keep track of two Tabu lists, one for the virtual nodes and another for the physical nodes. We fix the size of the Tabu list for the virtual nodes to be equal to 2, and that of the physical nodes to 7. Every time a move $m^v_{n,n'}$ is chosen from the set of candidate moves C, v is placed in the Tabu list of virtual nodes, and the source

host n is added to the Tabu list of physical nodes. By marking both v and n as Tabu, this prevents the selection of a future move that will place v back on n for the next x iterations (throughout our numerical results we let x be 7), thus reversing the move just made.

4. **Aspiration and Diversification Strategies:** The aspiration criteria consists of freeing a move from its Tabu status if it returns a solution with a cost lower than the best known solution so far. Finally, we set two main diversification strategies : a "random restart" and "penalizing moves with high frequency". The random restart is launched when no candidate moves can be found at a given iteration. The second diversification strategy consists of penalizing moves with a move frequency higher than x ($x = 7$ in our numerical results); that is moves that have been selected as the best candidate move for x number of iterations. The goal of these diversification strategies it to allow Tabu to leap into the unexplored search space.

Our Tabu's stopping criteria is set as the number of k consecutive iterations with no improvement. Once the stopping criteria is met, Tabu returns a node mapping solution for the VN in question. Next, the link mapping phase is launched using a shortest path algorithm (e.g., Dijkstra). It could happen that the best node mapping solution obtained by Tabu yields an infeasible link mapping solution. In this regard, we let Tabu keep track of x best solutions found (x is set to 7 in our experimental results). Assume Tabu executes at most I iterations, then the worst-case complexity of our proposed Tabu-based search is $\mathrm{O}(I.|V|.|N^2|)$.